

CTR Graphics Primer

Version 0.8

**The content of this document is highly confidential
and should be handled accordingly.**

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

1	Introduction	12
1.1	About This Document	12
1.2	Target Audience	12
1.3	Document Structure	12
2	CTR Graphics Characteristics	14
2.1	CTR Screens	14
2.2	Graphics Pipeline	15
2.2.1	Fragment Shader Effects	15
3	Vertex Processing	18
3.1	Vertex Attributes	18
3.2	Primitives.....	18
3.3	Vertex Shader Processes	19
3.3.1	Coordinate Conversion.....	20
3.3.2	Texture Coordinate Generation	21
3.3.3	Vertex Color Generation.....	21
3.3.4	Output to the Fragment Shader	22
3.4	Geometry Generation	23
3.4.1	Subdivision	23
3.4.2	Silhouettes.....	26
3.4.3	Particle System	29
4	Fragment Lighting	37
4.1	What Is Fragment Lighting?.....	37
4.2	Fragment Lighting Overview.....	39
4.3	Light Settings	41
4.3.1	Number of Lights	41
4.3.2	Position.....	42
4.3.3	Spotlight Direction	43
4.3.4	Light Colors	43
4.3.5	Distance Attenuation	43
4.3.6	Spotlight Attenuation.....	45
4.3.7	Geometric Factors.....	46
4.4	Material Settings	47
4.4.1	Material Colors	47
4.4.2	Global Ambient Color	47
4.4.3	Distribution (Specular Shape)	48
4.4.4	Reflection (Reflected Colors)	49
4.4.5	Fresnel (Transparency)	50

4.5	Lighting Formulas	51
4.5.1	Primary Color	52
4.5.2	Secondary Color	53
4.5.3	Alpha	54
4.6	Effects That Use Lookup Tables	54
4.6.1	Lookup Table Input Values	55
4.7	Layer Configurations	64
5	Textures	65
5.1	Texture Size	65
5.2	Texture Formats	65
5.2.1	Types of Texture Formats	66
5.2.2	Image List for Texture Formats	70
5.2.3	How to Choose a Texture Format	73
5.2.4	Texture Format Size Comparison	74
5.3	Cube Map Textures	75
5.4	Tiling Textures	77
5.5	Mipmaps	78
5.5.1	Mipmap Levels	78
5.5.2	LOD Bias	79
5.6	Texture Filters	80
5.7	Texture Mapping	83
5.7.1	UV Mapping	83
5.7.2	Projective Mapping	85
5.7.3	Cube Environment Mapping	86
5.7.4	Normal Mapping	87
5.8	Procedural Textures	88
5.8.1	Configuring Procedural Textures	89
5.8.2	Basic Shape	89
5.8.3	Noise	93
5.8.4	Tiling	96
5.8.5	Mipmap	99
5.8.6	Filter	99
5.8.7	Color	99
5.8.8	Comparing with Normal Textures	100
5.9	Usable Texture Combinations	101
6	Texture Combiners	102
6.1	What Are Texture Combiners?	102
6.2	Texture Combiner Structure	102
6.3	Texture Combiner Settings	103

6.3.1	Input Sources	104
6.3.2	Operands	105
6.3.3	Combiner Functions	106
6.3.4	Scale	107
6.3.5	Constant Colors	107
6.3.6	Combiner Buffers	107
7	Fragment Operations	109
7.1	Overview of Fragment Operations	109
7.2	Fog	110
7.3	Write Tests	112
7.3.1	Alpha Test	112
7.3.2	Depth Test	114
7.3.3	Stencil Test	115
7.4	Writing to the Framebuffer	117
7.4.1	Blending	117
7.4.2	Logical Operations	119
8	Framebuffers	121
8.1	Framebuffer Formats	121
8.2	LCD Output	122
8.3	Antialiasing	122
8.4	Application to Textures	123
9	Special Effects	124
9.1	Gas	124
9.2	Shadows	124
10	Samples	125
10.1	Sample Settings for Fragment Lighting	125
10.1.1	Lambert	125
10.1.2	Phong	128
10.1.3	Toon	132
10.1.4	Two-Layer Paint	135
10.1.5	Fresnel Reflection	139
10.2	Sample Settings for Procedural Textures	142
10.2.1	Water Surfaces	143
10.2.2	Fire	145
10.3	Sample Settings for Texture Combiners	147
10.3.1	Addition	148
10.3.2	Multiplication	150
10.3.3	Subtraction	152
10.3.4	Decals	154

10.3.5	Blending by an Arbitrary Ratio	156
10.3.6	Two-Color Interpolation.....	158
10.3.7	Applied Two-Color Interpolation.....	160
Appendix A.....		165
A.1	Hardware Configuration.....	165
A.1.1	CPU.....	165
A.1.2	GPU	165
A.1.3	Main Memory	166
A.1.4	VRAM.....	166
A.2	Overall Image of the Graphics Pipeline.....	166
A.3	Performance	168
A.3.1	Vertex Processing	168
A.3.2	Fragment Lighting	168
A.3.3	Textures	168
A.3.4	Texture Combiners.....	169
A.3.5	Fragment Operations	169

Tables

Table 3-1	Vertex Attributes Output to the Fragment Shader	22
Table 3-2	Geometry Generation	23
Table 3-3	Subdivision	24
Table 3-4	Subdivision Methods	25
Table 3-5	Silhouettes	26
Table 3-6	Edges.....	27
Table 3-7	Open Edges.....	27
Table 3-8	Particle System Settings.....	29
Table 3-9	Overall Particle Settings	30
Table 3-10	Individual Control Point Settings	34
Table 3-11	Particle Texture Coordinates.....	36
Table 4-1	Light Colors	43
Table 4-2	Material Colors	47
Table 4-3	Icons Used in Lighting Formulas	51
Table 4-4	Settings That Use Lookup Tables.....	54
Table 4-5	Lookup Table Input Values	55
Table 4-6	Settings Using Lookup Tables and Input Value Restrictions	55
Table 4-7	Layer Configurations	64
Table 5-1	Types of Texture Formats	66
Table 5-2	RGB Formats.....	67
Table 5-3	RGBA Formats.....	67
Table 5-4	Alpha Formats.....	68

Table 5-5 Luminance Formats	68
Table 5-6 Luminance Alpha Formats	69
Table 5-7 ETC Format.....	69
Table 5-8 ETCA4 Format	70
Table 5-9 HILO.....	70
Table 5-10 RGB Image List for Texture Formats	70
Table 5-11 Alpha Image List for Texture Formats	72
Table 5-12 Tiling Textures	77
Table 5-13 Texture Filters.....	80
Table 5-14 Texture Filtering (with Mipmaps in Use).....	82
Table 5-15 Basic Shape (G Function).....	90
Table 5-16 Control Parameters.....	93
Table 5-17 Repetition Methods	96
Table 5-18 Features of Procedural and Normal Textures	100
Table 5-19 Usable Texture Combinations	101
Table 5-20 Combinations of Textures and Texture Coordinates	101
Table 6-1 Texture Combiner Settings.....	103
Table 6-2 Input Sources.....	104
Table 6-3 Color Operands.....	105
Table 6-4 Alpha Operands	105
Table 6-5 Combiner Functions.....	106
Table 7-1 Fog Settings	110
Table 7-2 Write Tests.....	112
Table 7-3 Alpha Test Settings.....	113
Table 7-4 Comparison Functions for the Alpha Test	113
Table 7-5 Comparison Functions for the Depth Test.....	114
Table 7-6 Stencil Test Settings.....	115
Table 7-7 Comparison Functions for the Stencil Test	116
Table 7-8 Blend Settings.....	118
Table 7-9 Source and Destination Factors.....	118
Table 7-10 Blending Equations	119
Table 7-11 Logical Operations	120
Table 8-1 Framebuffer Structure	121
Table 8-2 Buffer Format	121
Table 8-3 Antialiasing	122
Table 8-4 Application to Textures	123
Table 10-1 Sample Settings for the Lambert Shading Model (Primary Color).....	126
Table 10-2 Sample Settings for the Phong Shading Model (Primary Color)	129
Table 10-3 Sample Settings for the Phong Shading Model (Secondary Color).....	129
Table 10-4 Sample Settings for the Toon Shading Model (Secondary Color)	133
Table 10-5 Sample Settings for the Two-Layer Paint Model (Primary Color)	136
Table 10-6 Sample Settings for the Two-Layer Paint Model (Secondary Color)	136

Table 10-7 Sample Settings for Fresnel Reflection (Primary Color)	140
Table 10-8 Fresnel Reflection Texture Combiner Settings	140
Table 10-9 Basic Procedural Texture Settings.....	142
Table 10-10 Sample Settings for the Surface of the Water	144
Table 10-11 Sample Settings for Fire	145
Table 10-12 Settings for Combiners That Have Not Been Explicitly Configured	147
Table 10-13 Settings for Addition.....	149
Table 10-14 Settings for Multiplication.....	151
Table 10-15 Settings for Subtraction	153
Table 10-16 Settings for Decals	155
Table 10-17 Settings Used to Blend by a Ratio.....	157
Table 10-18 Two-Color Interpolation Settings	159
Table 10-19 Settings for Applying Two-Color Interpolation	163

Figures

Figure 1-1 Document Structure	13
Figure 2-1 CTR Screens	14
Figure 2-2 Graphics Pipeline.....	15
Figure 2-3 Fragment Lighting	16
Figure 2-4 Normal Mapping.....	16
Figure 2-5 Cube Environment Mapping	17
Figure 2-6 Procedural Textures	17
Figure 3-1 Vertex Attributes.....	18
Figure 3-2 Primitive Types.....	18
Figure 3-3 Consecutive Triangle Methods	19
Figure 3-4 Front Face of a Polygon.....	19
Figure 3-5 Coordinate Conversion	20
Figure 3-6 Texture Coordinate Generation.....	21
Figure 3-7 Vertex Color Generation	22
Figure 3-8 Geometry Generation	23
Figure 3-9 Subdivision.....	24
Figure 3-10 Levels.....	24
Figure 3-11 Subdivision Methods	25
Figure 3-12 Silhouettes	26
Figure 3-13 Edges.....	27
Figure 3-14 Open Edges	28
Figure 3-15 Silhouette Methods	28
Figure 3-16 Example Particles	29
Figure 3-17 Trajectory Settings	30
Figure 3-18 Particle Positions	31
Figure 3-19 Overall Particle Settings (Time and Speed).....	32

Figure 3-20 Overall Particle Settings (Particle Size)	33
Figure 3-21 Control Point Settings.....	35
Figure 3-22 Individual Control Point Settings (Texture Coordinates)	36
Figure 4-1 Vertex Lighting and Fragment Lighting	38
Figure 4-2 Choice of Lighting Methods.....	39
Figure 4-3 Fragment Lighting Overview	40
Figure 4-4 Light Settings.....	41
Figure 4-5 Types of Light Sources.....	41
Figure 4-6 Light Source Depictions Due to Light Positions	42
Figure 4-7 Spotlight Direction	43
Figure 4-8 Distance Attenuation	44
Figure 4-9 Spotlight Attenuation.....	45
Figure 4-10 Geometric Factors.....	46
Figure 4-11 Material Settings.....	47
Figure 4-12 Distribution.....	48
Figure 4-13 Reflection.....	49
Figure 4-14 Fresnel.....	50
Figure 4-15 Primary Color Formula	52
Figure 4-16 Secondary Color Formula	53
Figure 4-17 Alpha Formula	54
Figure 4-18 Lookup Table Structure.....	55
Figure 4-19 Relationship Between the Input Value and the Angle Created by Two Vectors	56
Figure 4-20 Input Values and Base Vectors	57
Figure 4-21 LN Input Value	58
Figure 4-22 NV Input Value.....	59
Figure 4-23 NH Input Value	60
Figure 4-24 VH Input Value.....	61
Figure 4-25 SP Input Value	62
Figure 4-26 CP Input Value.....	63
Figure 5-1 Texture Size.....	65
Figure 5-2 How to Choose a Texture Format.....	73
Figure 5-3 How to Choose a Texture Format for Normal Mapping.....	74
Figure 5-4 Texture Format Size Comparison (for a 256x256 Texture)	75
Figure 5-5 Cube Map Textures	76
Figure 5-6 Tiling Textures.....	77
Figure 5-7 Sample Application of Mipmaps	78
Figure 5-8 Example of Using Mipmaps to Remove Moiré Patterns	78
Figure 5-9 Mipmap Levels	79
Figure 5-10 LOD Bias	80
Figure 5-11 Point Sampling and Bilinear Filtering	81
Figure 5-12 Texture Coordinate System	83
Figure 5-13 UV Mapping.....	84

Figure 5-14 Projective Mapping	85
Figure 5-15 Cube Environment Mapping	86
Figure 5-16 Normal Mapping.....	87
Figure 5-17 Procedural Textures	88
Figure 5-18 Procedural Texture Examples	88
Figure 5-19 Configuring Procedural Textures.....	89
Figure 5-20 Basic Shape	89
Figure 5-21 Basic Shape (F Function)	92
Figure 5-22 Noise	93
Figure 5-23 Noise Control Parameters.....	94
Figure 5-24 Noise Functions	95
Figure 5-25 Shift	98
Figure 5-26 Shift	99
Figure 5-27 Filter	99
Figure 5-28 Color.....	100
Figure 6-1 What Are Texture Combiners?	102
Figure 6-2 Texture Combiner Structure	103
Figure 6-3 Texture Combiner Settings.....	104
Figure 6-4 Constant Colors	107
Figure 6-5 Combiner Buffer 0	107
Figure 6-6 Combiner Buffers 1–4	108
Figure 6-7 Sample Use of Combiner Buffers	108
Figure 7-1 Overview of Fragment Operations.....	109
Figure 7-2 Fog Lookup Table and Equation	110
Figure 7-3 Fog	111
Figure 7-4 Alpha Test.....	112
Figure 7-5 Sample Usage of the Alpha Test.....	113
Figure 7-6 Depth Test.....	114
Figure 7-7 Sample Usage of the Depth Test	115
Figure 7-8 Stencil Test.....	115
Figure 7-9 Sample Usage of the Stencil Test	116
Figure 7-10 Blending	117
Figure 7-11 Logical Operations	119
Figure 8-1 LCD Output	122
Figure 8-2 Antialiasing	123
Figure 10-1 Example of Lambert Shading	125
Figure 10-2 Settings for the Lambert Shading Model	126
Figure 10-3 Formula for Lambert Shading (Primary Color).....	127
Figure 10-4 Example of Phong Shading	128
Figure 10-5 Settings for the Phong Shading Model	128
Figure 10-6 Formula for Phong Shading (Primary Color)	130
Figure 10-7 Formula for Phong Shading (Secondary Color)	131

Figure 10-8 Example of Toon Shading	132
Figure 10-9 Settings for the Toon Shading Model	132
Figure 10-10 Formula for Toon Shading (Secondary Color).....	134
Figure 10-11 Example of Two-Layer Paint.....	135
Figure 10-12 Settings for the Two-Layer Paint Model	135
Figure 10-13 Formula for Two-Layer Paint (Primary Color)	137
Figure 10-14 Formula for Two-Layer Paint (Secondary Color).....	138
Figure 10-15 Fresnel Reflection	139
Figure 10-16 Fresnel Reflection Settings	139
Figure 10-17 Fresnel Reflection Algorithm (Primary Color).....	141
Figure 10-18 Water Surface.....	143
Figure 10-19 Sample Settings for Fire.....	145
Figure 10-20 Sample Addition	148
Figure 10-21 Processing for Addition.....	149
Figure 10-22 Sample Multiplication	150
Figure 10-23 Processing for Multiplication	151
Figure 10-24 Sample Subtraction	152
Figure 10-25 Processing for Subtraction	153
Figure 10-26 Sample Decals	154
Figure 10-27 Processing for Decals	155
Figure 10-28 Example of Blending by a Ratio	156
Figure 10-29 Process of Blending by a Ratio	157
Figure 10-30 Two-Color Interpolation	158
Figure 10-31 Processing Two-Color Interpolation	159
Figure 10-32 Applied Two-Color Interpolation	161
Figure 10-33 Processing the Applied Two-Color Interpolation	162
Figure 10-34 Examples With Changed Constant Colors.....	164
Figure A-1 Hardware Configuration	165
Figure A-2 Overall Image of the Graphics Pipeline	167

1 Introduction

1.1 About This Document

The purpose of this document is to explain the main CTR graphics features required to develop CTR software.

To make it easier to understand CTR graphics, descriptions of some features have been omitted along with programming restrictions and other information. For more details on each of these features, see the programming manuals and other documentation.

Note: This document is based on the most recent CTR graphics specifications. Note that certain features are not supported in some development environments.

We are still preparing some of the figures that appear in this version of the document.

1.2 Target Audience

This document is intended for the following people who develop CTR software.

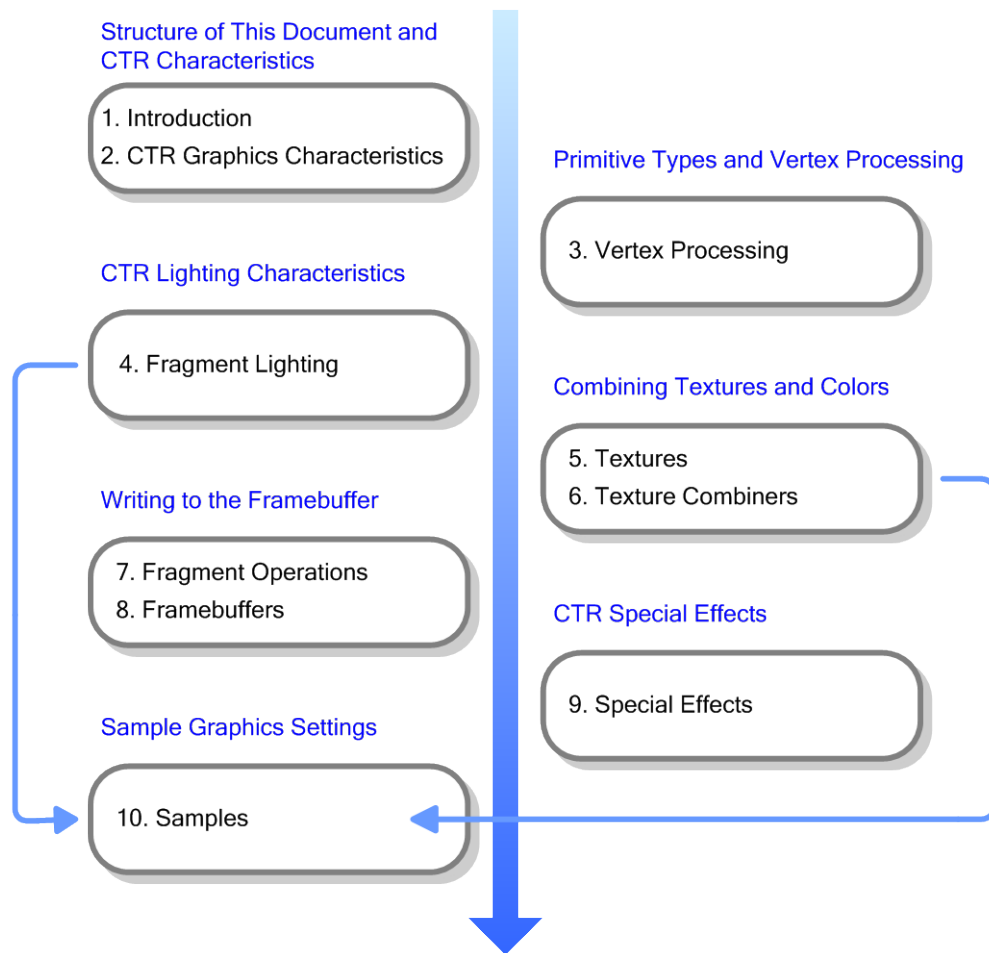
- Designers
- Programmers
- Planners
- Directors

You need a basic understanding of 3D graphics to continue reading this document.

1.3 Document Structure

This document comprises the following chapters. The flow of graphics processing is explained first, followed by concrete examples.

By simply reading through this document, you can deepen your overall understanding of graphics. You can also refer to each section as necessary for more details.

Figure 1-1 Document Structure

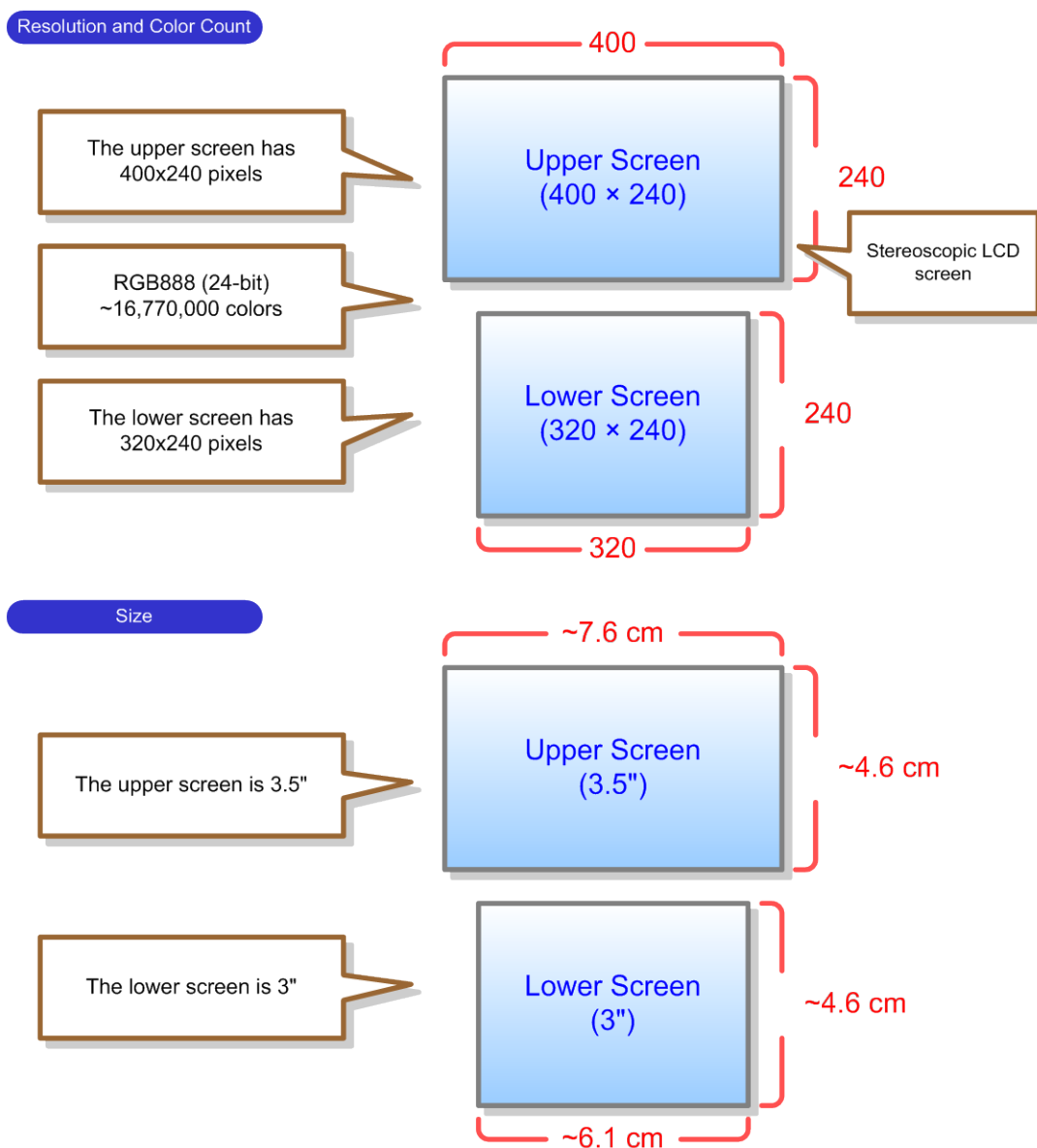
2 CTR Graphics Characteristics

2.1 CTR Screens

The CTR system has two screens: an upper and lower LCD. The upper screen is an autostereoscopic LCD. In stereoscopic display mode, the two pixels for the left and right eyes are shown as large as a single pixel in normal display mode. Stereoscopic display requires a 400x240 image for each eye.

The following figure shows the screen resolution, pixel count, and size.

Figure 2-1 CTR Screens

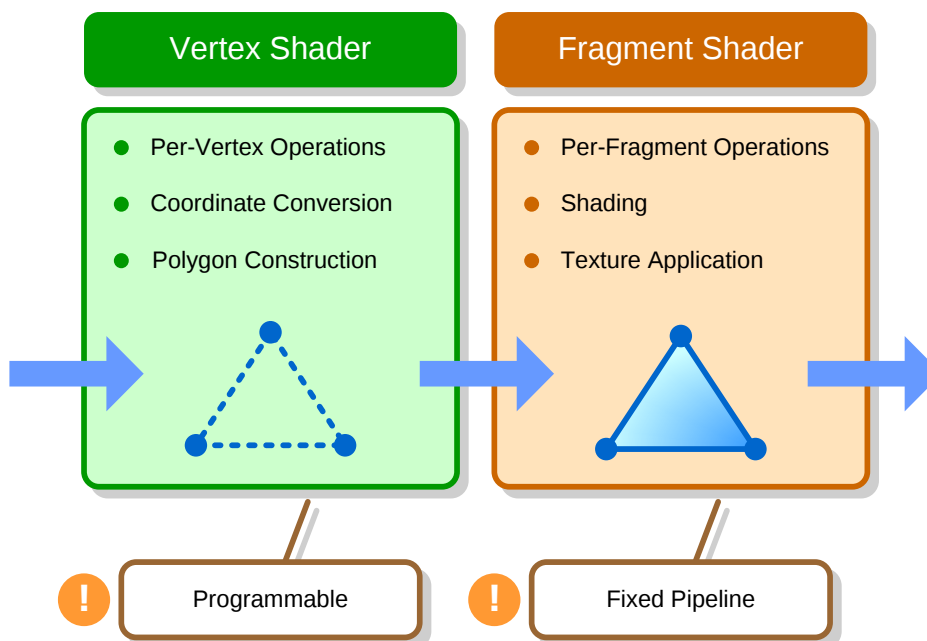


2.2 Graphics Pipeline

The CTR graphics pipeline has the following two characteristics:

- A programmable vertex shader capable of any vertex processing.
- A (fixed-pipeline) fragment shader capable of various visual effects.

Figure 2-2 Graphics Pipeline



The vertex shader is responsible for per-vertex operations. Among other things, it transforms and generates coordinates and creates polygons.

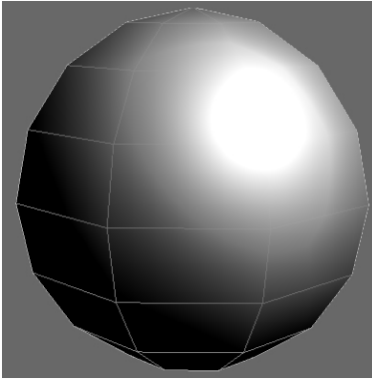
The fragment shader is responsible for per-fragment (per-pixel) operations. Among other things, it applies lighting and textures and blends colors.

2.2.1 Fragment Shader Effects

The fragment shader can generate the following visual effects on a CTR system.

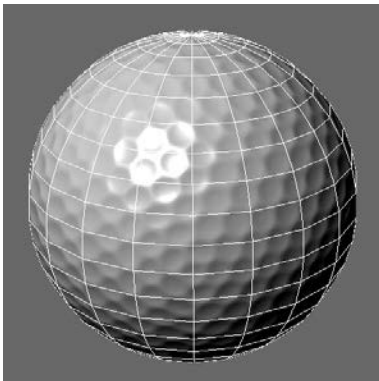
2.2.1.1 Fragment Lighting

This lights individual fragments. It is explained in Chapter 4 Fragment Lighting.

Figure 2-3 Fragment Lighting

2.2.1.2 Normal Mapping

This can emulate small bumps on the surface of an object. It is explained in section 5.7.4 Normal Mapping.

Figure 2-4 Normal Mapping

2.2.1.3 Cube Environment Mapping

This can reflect a scene onto an object placed in it. It is explained in section 5.7.3 Cube Environment Mapping.

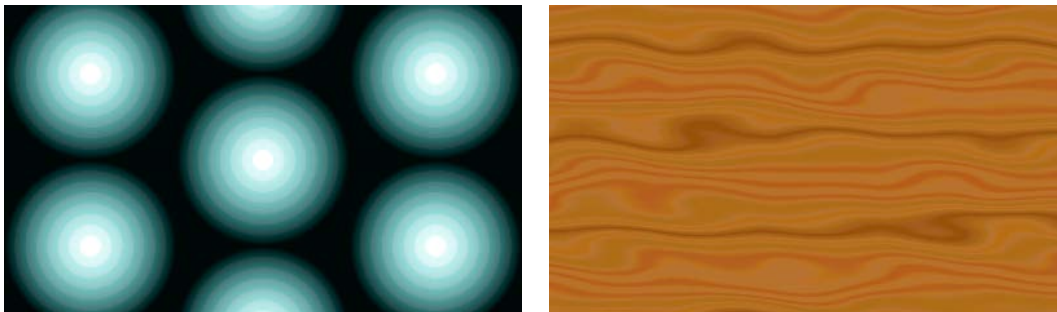
Figure 2-5 Cube Environment Mapping



2.2.1.4 Procedural Textures

This can automatically generate textures according to configured parameters. It is explained in section 5.8 Procedural Textures.

Figure 2-6 Procedural Textures

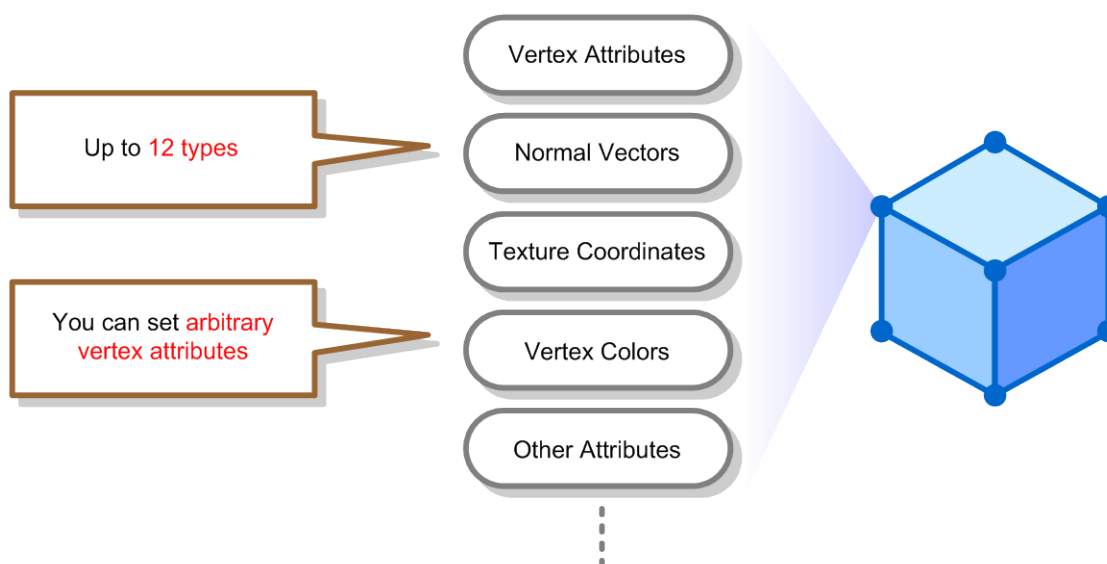


3 Vertex Processing

3.1 Vertex Attributes

Data that can be set for each vertex, such as vertex coordinates and normal vectors, are called *vertex attributes* for CTR. You can set up to 12 types of vertex attributes for a single vertex.

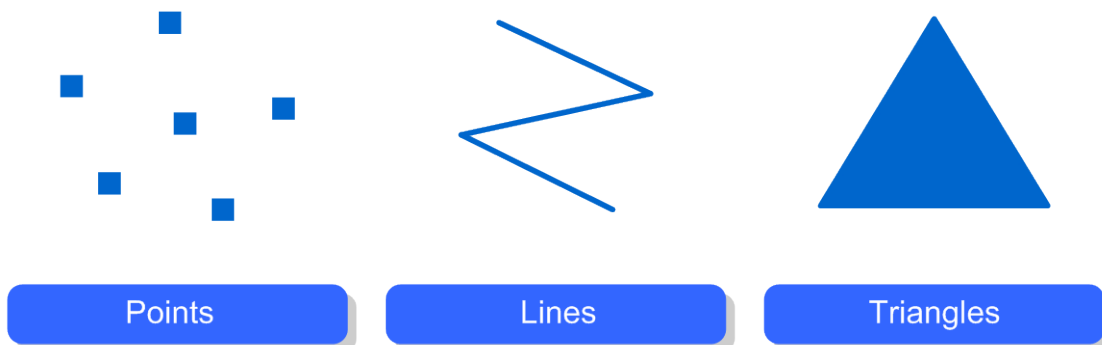
Figure 3-1 Vertex Attributes



3.2 Primitives

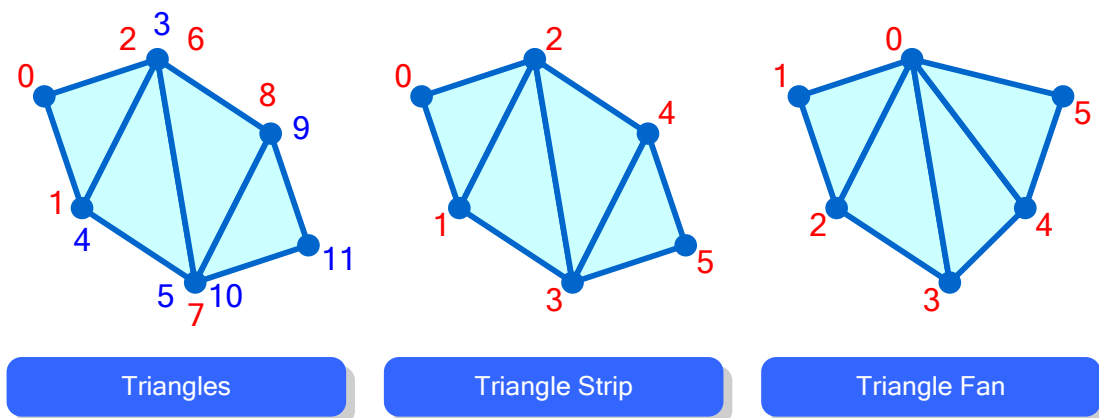
The CTR system allows you to use three types of primitives.

Figure 3-2 Primitive Types



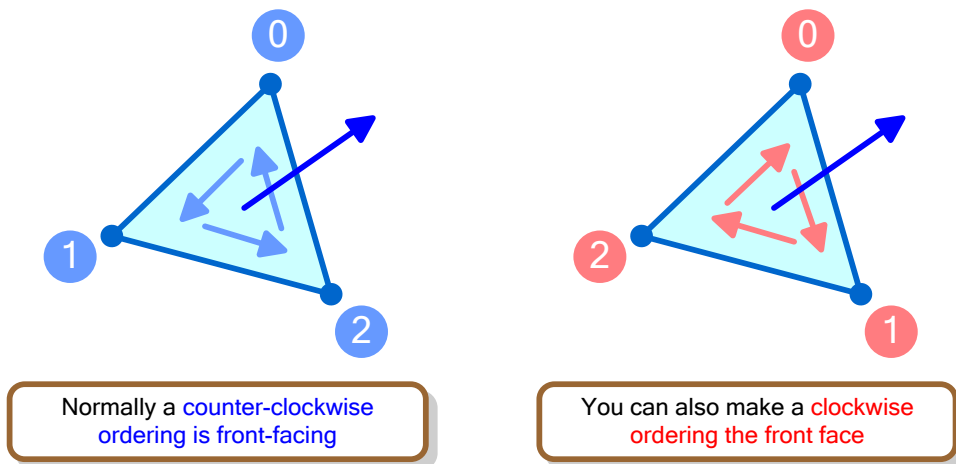
Consecutive triangle vertices can be specified in the following three ways.

Figure 3-3 Consecutive Triangle Methods



A polygon has a front and a back face that are determined by the order in which its vertices are connected. The front face of a polygon is normally determined using a counterclockwise order but this can be changed.

Figure 3-4 Front Face of a Polygon



3.3 Vertex Shader Processes

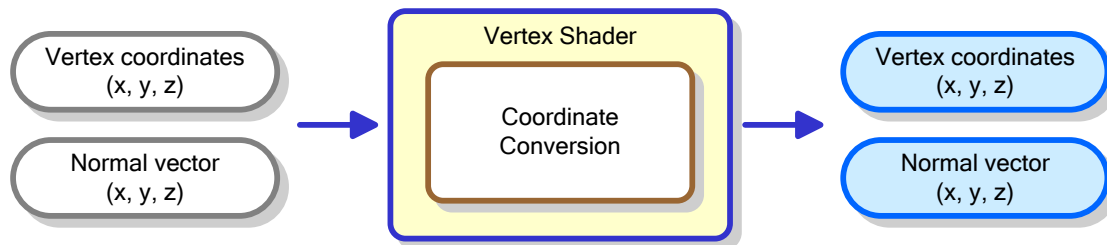
The vertex shader can run the following processes on input vertex attributes. Data that is converted or generated by the vertex shader is output to the fragment shader.

- Coordinate conversion
- Texture coordinate generation
- Vertex color generation

3.3.1 Coordinate Conversion

Input vertex attributes and normal vectors are converted into screen coordinates.

Figure 3-5 Coordinate Conversion

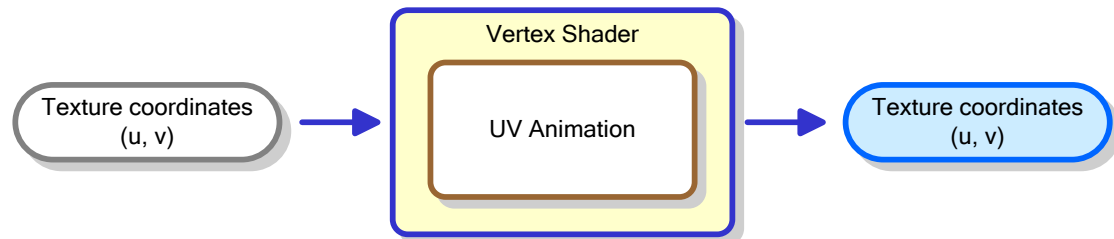


3.3.2 Texture Coordinate Generation

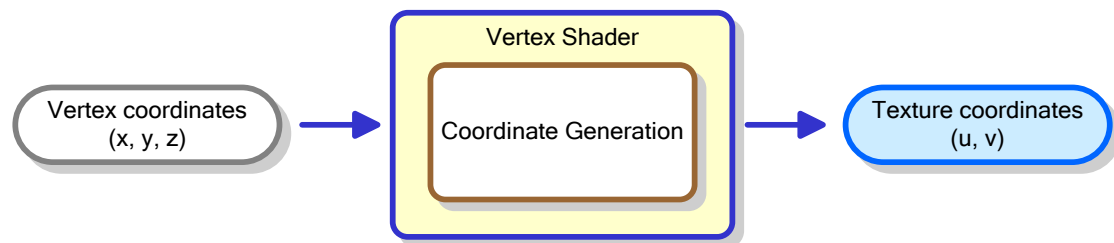
Texture coordinates for texture mapping are generated for each vertex.

Figure 3-6 Texture Coordinate Generation

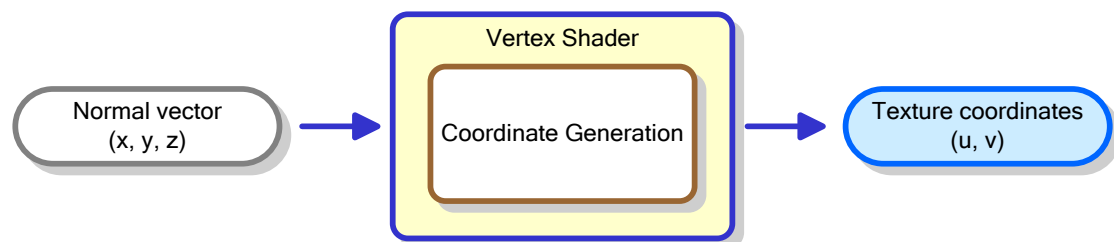
Example 1 Convert and then output texture coordinates input as vertex attributes



Example 2 Generate texture coordinates for perspective mapping from vertex coordinates



Example 3 Generate texture coordinates for environment mapping from normal vectors

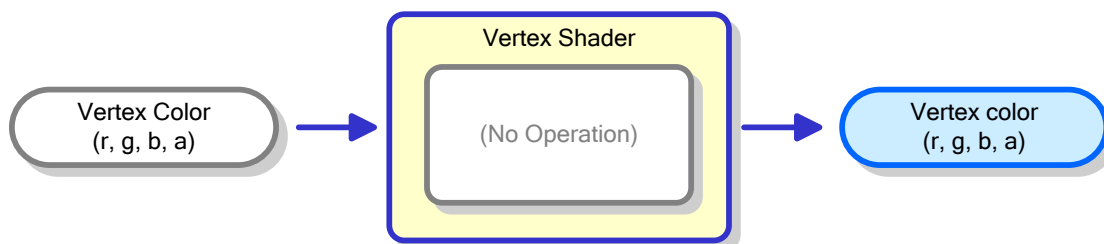


3.3.3 Vertex Color Generation

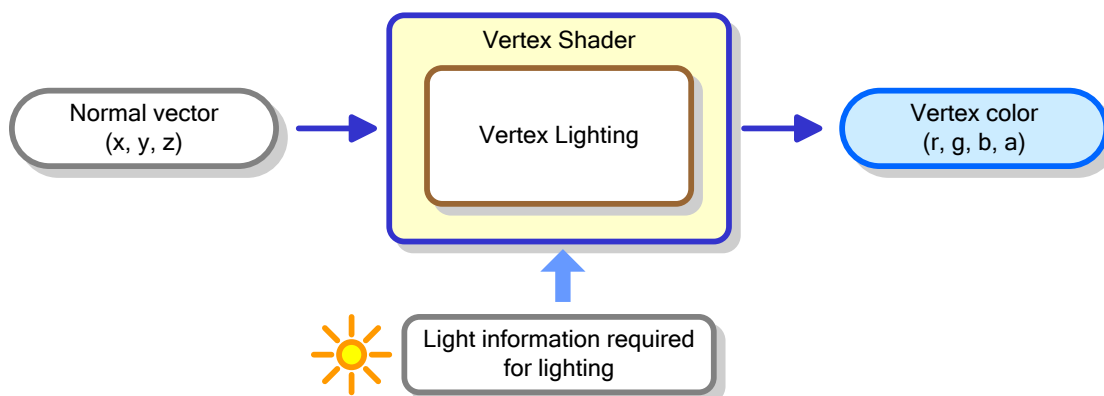
A vertex color is generated for each vertex.

Figure 3-7 Vertex Color Generation

Example 1 Output an unchanged vertex color that was input as a vertex attribute



Example 2 Apply vertex lighting in the vertex shader and output the resulting color



3.3.4 Output to the Fragment Shader

Vertex attributes processed by the vertex shader are output to the fragment shader. The following table shows the main vertex attributes output to the fragment shader.

Table 3-1 Vertex Attributes Output to the Fragment Shader

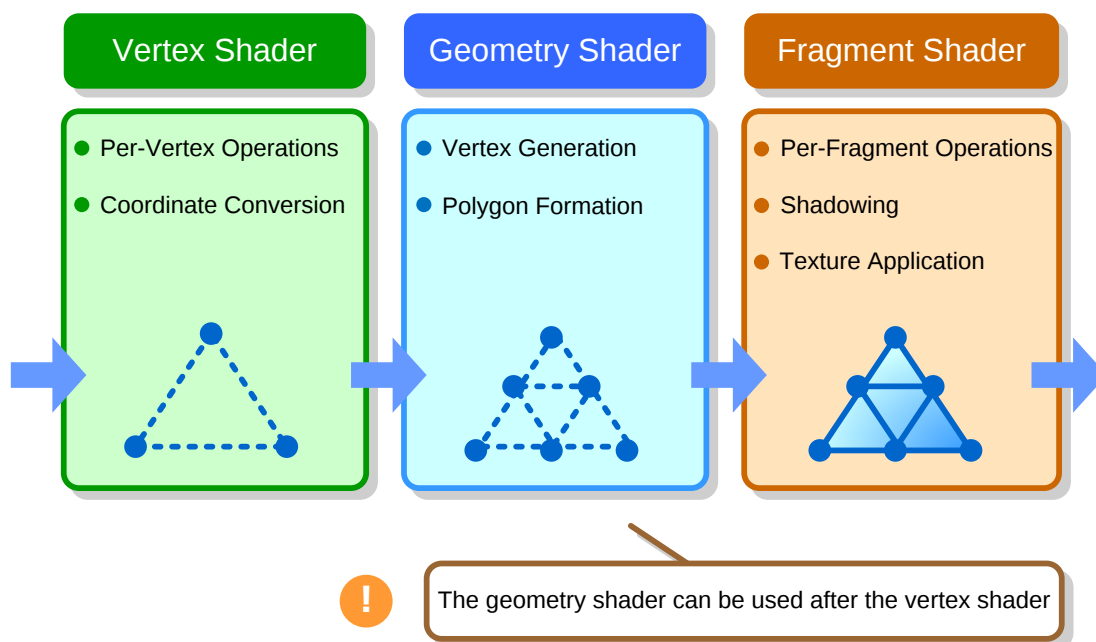
Output Attribute	Data
Vertex coordinates	x, y, z
Normal vector	x, y, z
Vertex color	r, g, b, a
Texture coordinate0	u, v
Texture coordinate 1	u, v
Texture coordinate 2	u, v

3.4 Geometry Generation

The CTR system has a feature called *geometry generation* that allows you to generate vertices based on output from the vertex shader. Geometry generation is run by the geometry shader, which is located between the vertex shader and the fragment shader.

Using geometry generation, you can represent smooth surfaces with a small number of vertices and create multiple particles around input vertices given as control points.

Figure 3-8 Geometry Generation



The CTR system has the following three features for generating geometry.

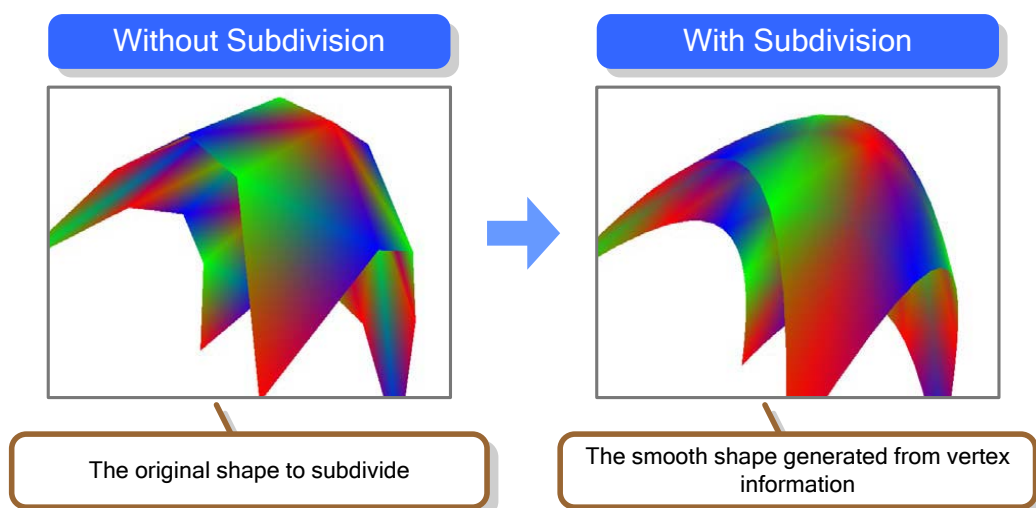
Table 3-2 Geometry Generation

Type of Geometry Generation	Description
Subdivision	Smooths an object's shape.
Silhouettes	Generates an outline around an object.
Particle system	Generates particles.

3.4.1 Subdivision

Subdivision refers to a feature that subdivides objects in order to represent smooth surfaces with a small number of vertices. By controlling the number of subdivisions, you can change an object's smoothness according to its distance from the camera.

Figure 3-9 Subdivision



There are two subdivision settings, as shown in the following table.

Table 3-3 Subdivision

Setting	Description
Level	Selects a level that indicates the fineness of the subdivision.
Subdivision method	Selects the subdivision method.

3.4.1.1 Levels

You can choose one of three values—0, 1, or 2—to indicate the smoothness of a subdivided shape (patch). Larger values cause subdivision to produce smoother shapes and a value of 0 causes subdivision to be skipped.

Figure 3-10 Levels



Note: The processing load on the geometry shader increases with the subdivision level.

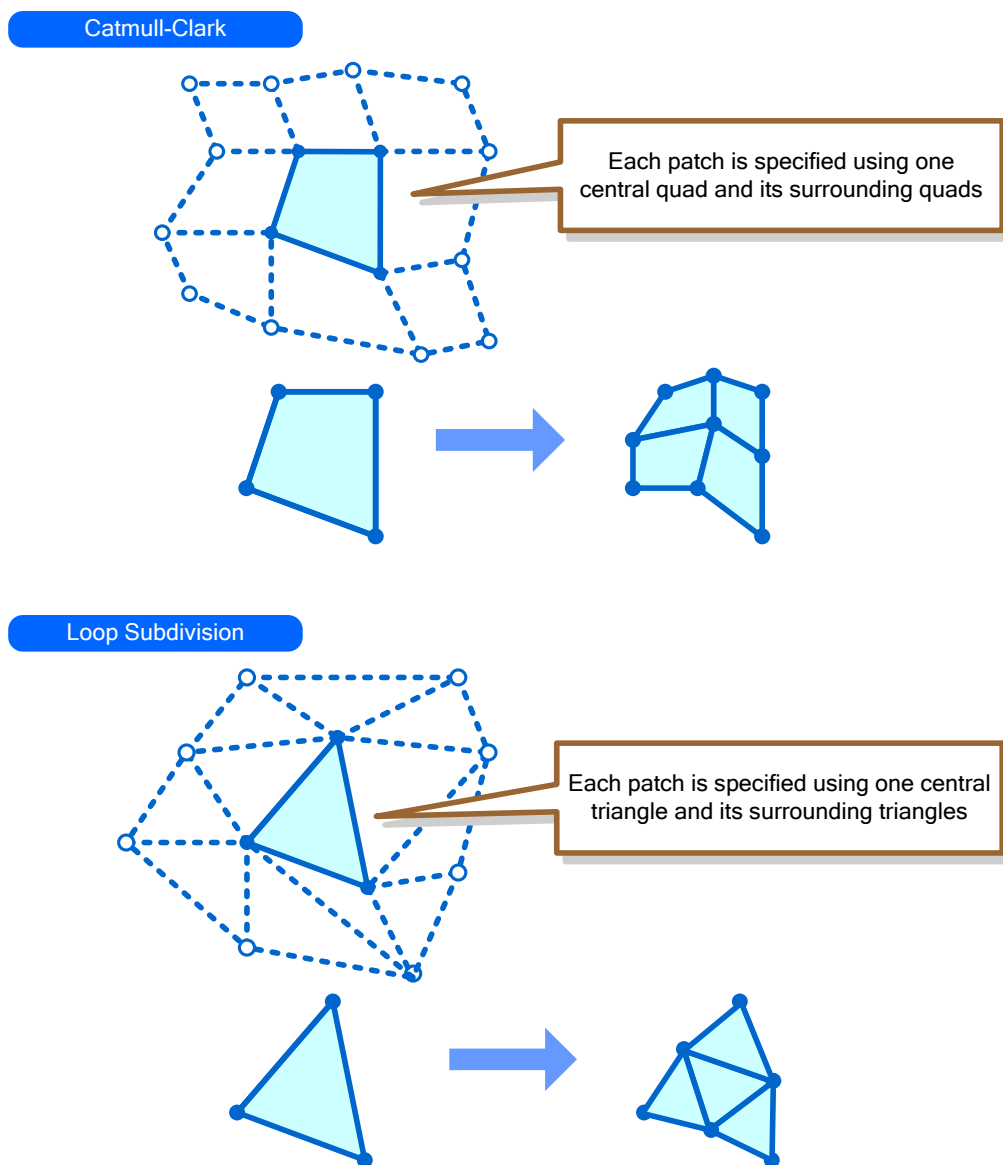
3.4.1.2 Subdivision Methods

You can choose from two subdivision methods: Catmull-Clark and Loop subdivision. Each method uses a different polygon shape for the *patches* that it subdivides. To subdivide the polygon that is in the center of a group of polygons, however, each method simultaneously requires information for the center polygon and for the polygons that surround it.

Table 3-4 Subdivision Methods

Method	Description
Catmull-Clark	Subdivides quadrilaterals.
Loop subdivision	Subdivides triangles.

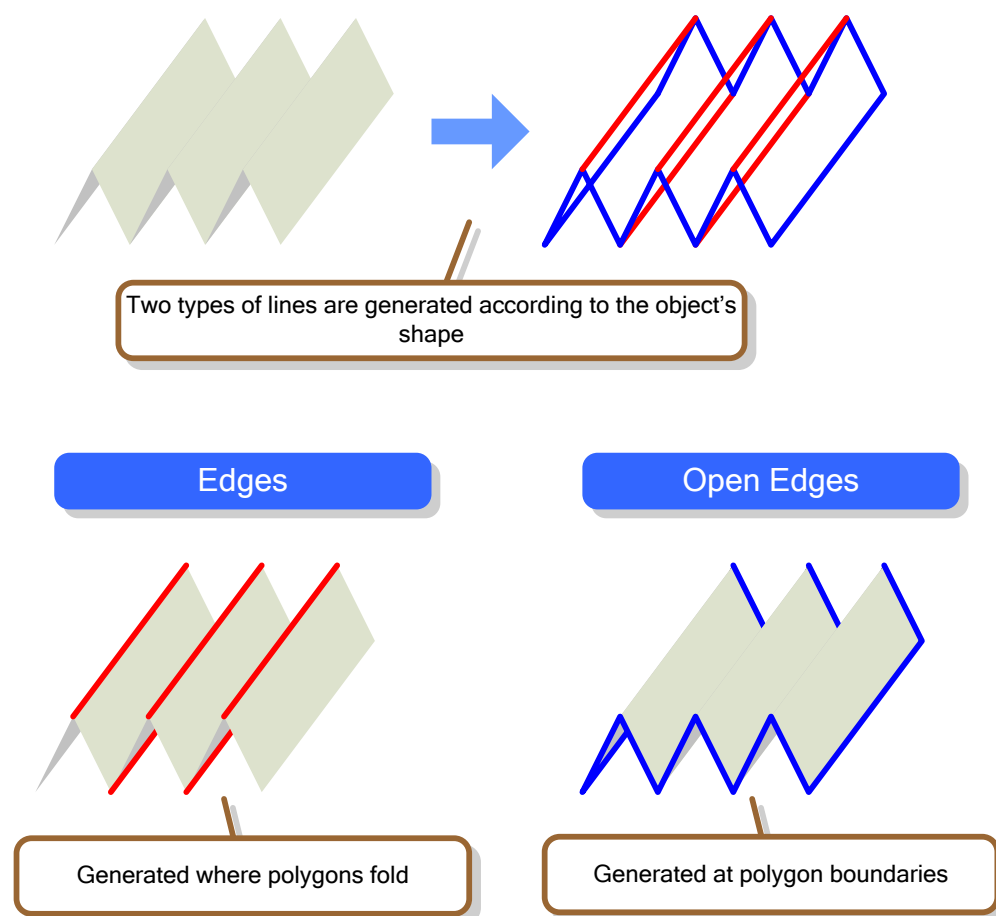
Figure 3-11 Subdivision Methods



3.4.2 Silhouettes

Silhouettes represent a feature for automatically generating an outline around an object. This outline comprises two types of lines—edges and open edges—that are determined by an object's shape.

Figure 3-12 Silhouettes



Note: Silhouettes cannot be lit because normal data is not generated for them.

Silhouette settings can be largely divided into the following three categories.

Table 3-5 Silhouettes

Setting	Description
Edges	Settings that affect the lines generated where polygons fold.
Open edges	Settings that affect the lines generated at polygon boundaries.
Silhouette method	Selects how the vertices used for silhouette generation are specified.

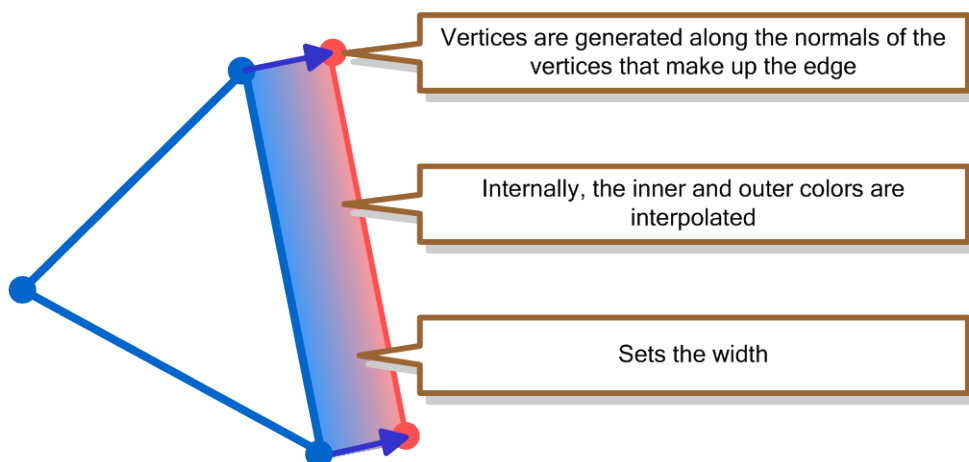
3.4.2.1 Edges

Edges are lines generated where polygons fold: on the sides of triangles that, when compared with other adjacent triangles, are determined to be a part of the outline. Edges use the following four settings.

Table 3-6 Edges

Setting	Description
Inner color	The vertex color is used.
Outer color	Sets the color on the lines' outer side.
Width	Sets the line width.
Scale	Configures whether the line width changes with the distance from the camera.

Figure 3-13 Edges



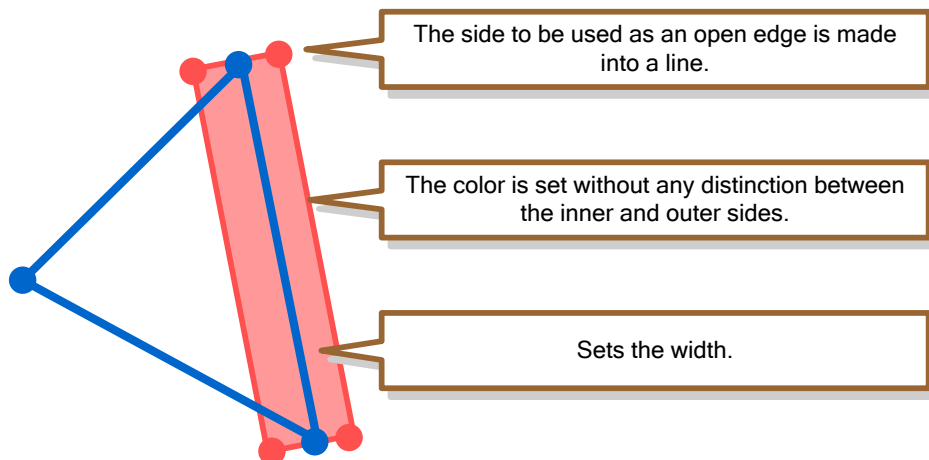
3.4.2.2 Open Edges

Open edges are lines generated at polygon boundaries: on the sides of triangles that have no adjacent triangles. Open edges use the following four settings.

Table 3-7 Open Edges

Setting	Description
Enable/Disable	Configures whether to generate open edges.
Color	Sets the line color.
Width	Sets the line width.
Scale	Configures whether the line width changes with the distance from the camera.

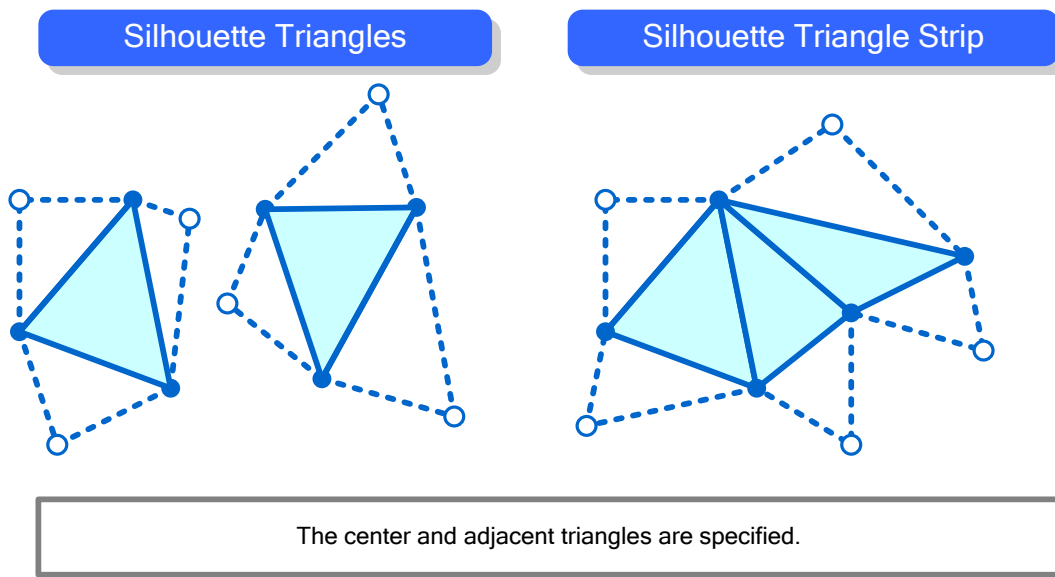
Figure 3-14 Open Edges



3.4.2.3 Silhouette Methods

There are two silhouette methods: *silhouette triangles*, in which each triangle is specified separately, and *silhouette triangle strips*, in which triangles are specified consecutively. To determine whether to generate silhouettes from information on adjacent triangles, however, each method simultaneously requires information for the center triangle and for the triangles that surround it.

Figure 3-15 Silhouette Methods



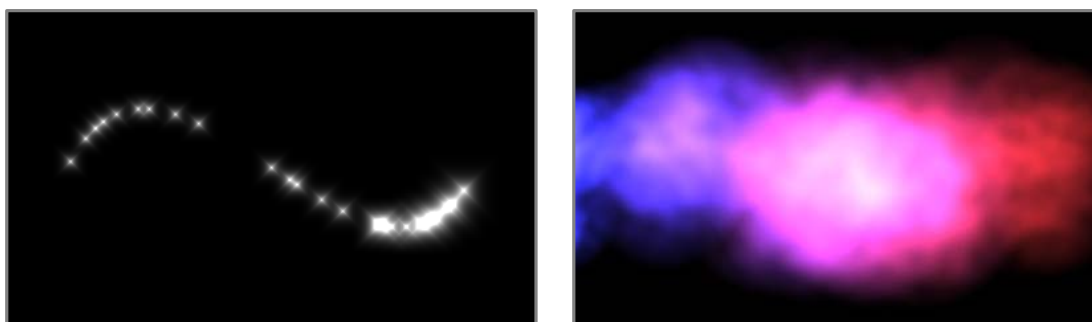
Note: Silhouette triangle strips are faster to process than silhouette triangles.

3.4.3 Particle System

The particle system feature automatically generates multiple polygons (particles).

Instead of sending specific information for each individual polygon, this sets parameters—such as the overall particle trajectory and size—to automatically generate an animation with multiple polygons. By applying textures to the generated polygons, you can use effects that are represented by fine particles such as snow and smoke.

Figure 3-16 Example Particles



Multiple polygons (particles) are automatically generated

Note: Every polygon generated by the particle system is a square (billboard) facing the camera. These cannot be lit because normal data is not generated.

Particle system settings are divided between the following three broad categories.

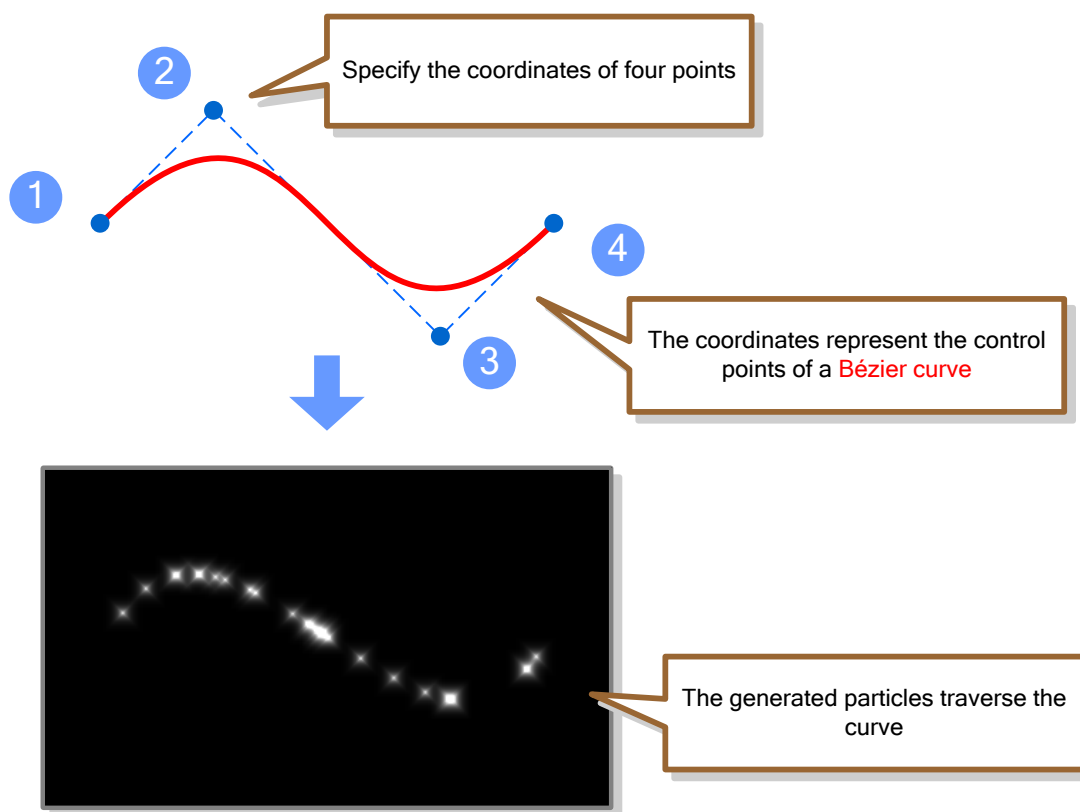
Table 3-8 Particle System Settings

Setting	Description
Trajectory settings	Sets a particle's trajectory using four coordinates.
Overall settings	Sets the number and time for particles overall.
Individual control point settings	Configures changes in particles' color and size.

3.4.3.1 Trajectory Settings

A particle system requires four input coordinates: these are the control points of a Bézier curve. Each particle is generated at the start of the Bézier curve and then follows the curve until it disappears at the end.

Figure 3-17 Trajectory Settings



Bézier Curve

A curve that is generated from four control points.
The first and fourth control points are respectively the starting and ending points of the curve.

3.4.3.2 Overall Settings

The following table shows the six settings for particles overall.

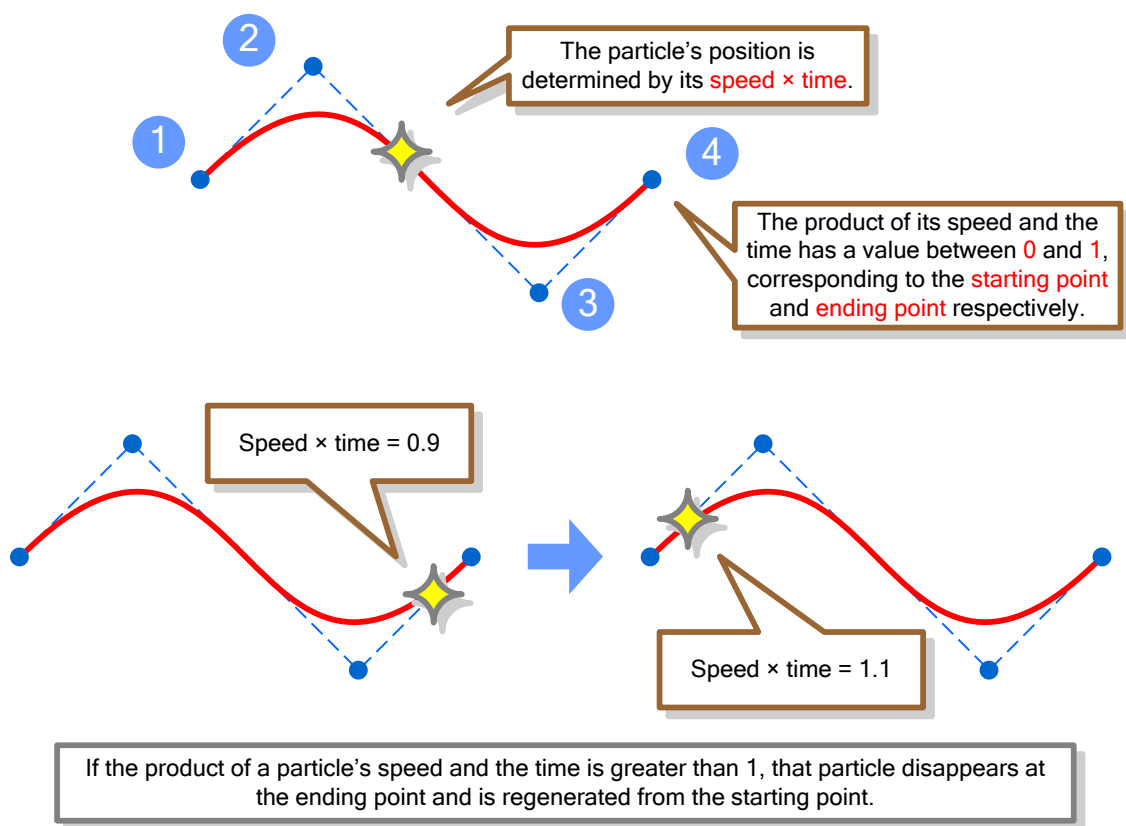
Table 3-9 Overall Particle Settings

Setting	Description
Time	Sets the current time of the overall particle system. You can animate particle motion by changing the time every frame.
Time clamping	Configures whether particles are generated for times that are not between 0 and 1.
Speed	Sets the speed of particle movement.
Generation count	Sets the number of particles (up to 255).

Setting	Description
Size variations due to distance	Configures how the distance from the camera changes the particle size (the perspective).
Minimum/maximum size	Sets the minimum and maximum particle sizes.

A particle's position on the curve is determined by the product of its speed and the time. Multiple particles are generated with the overall time changed randomly by 100–200%.

Figure 3-18 Particle Positions



Generating Multiple Particles

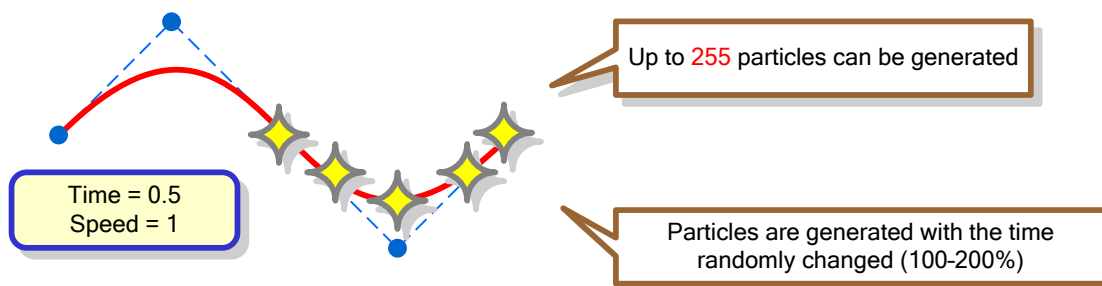


Figure 3-19 Overall Particle Settings (Time and Speed)

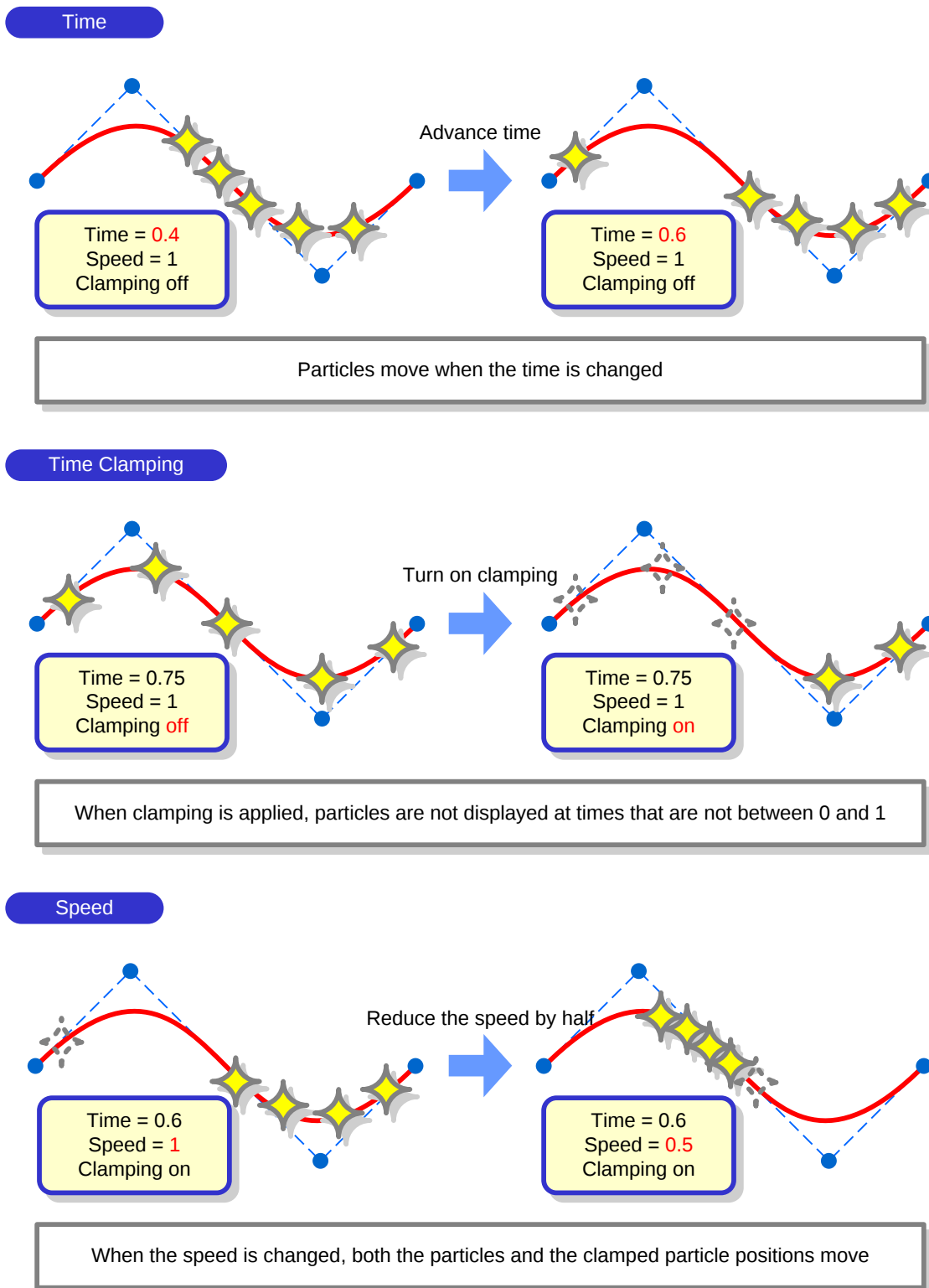
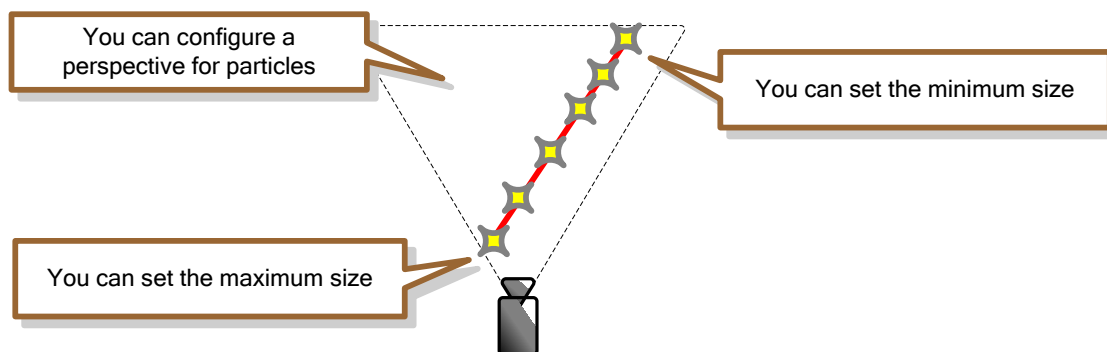
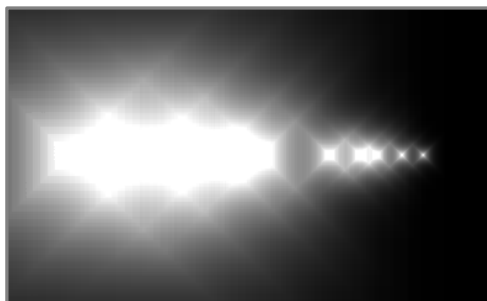
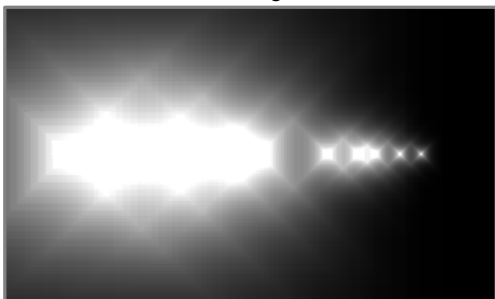
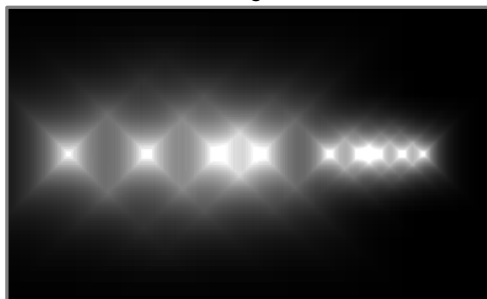


Figure 3-20 Overall Particle Settings (Particle Size)**Size Variations Due to Distance****Do Not Resize****Resize**

Configures how the distance from the camera changes the size (the perspective)

Minimum/Maximum Size**Not Configured****Configured**

Sets the minimum and maximum size

3.4.3.3 Individual Control Point Settings

The following five items can be set for each control point. Because settings are interpolated between each control point before being applied to particles moving along the curve, it is possible to gradually change a particle's size and color.

Table 3-10 Individual Control Point Settings

Setting	Description
Size	Sets a particle's size.
Color	Sets vertex colors.
Control point scattering	Sets a bounding box within which to scatter control points.
Texture rotation	Sets a texture's angle of rotation.
Texture scaling	Sets a texture's scale.

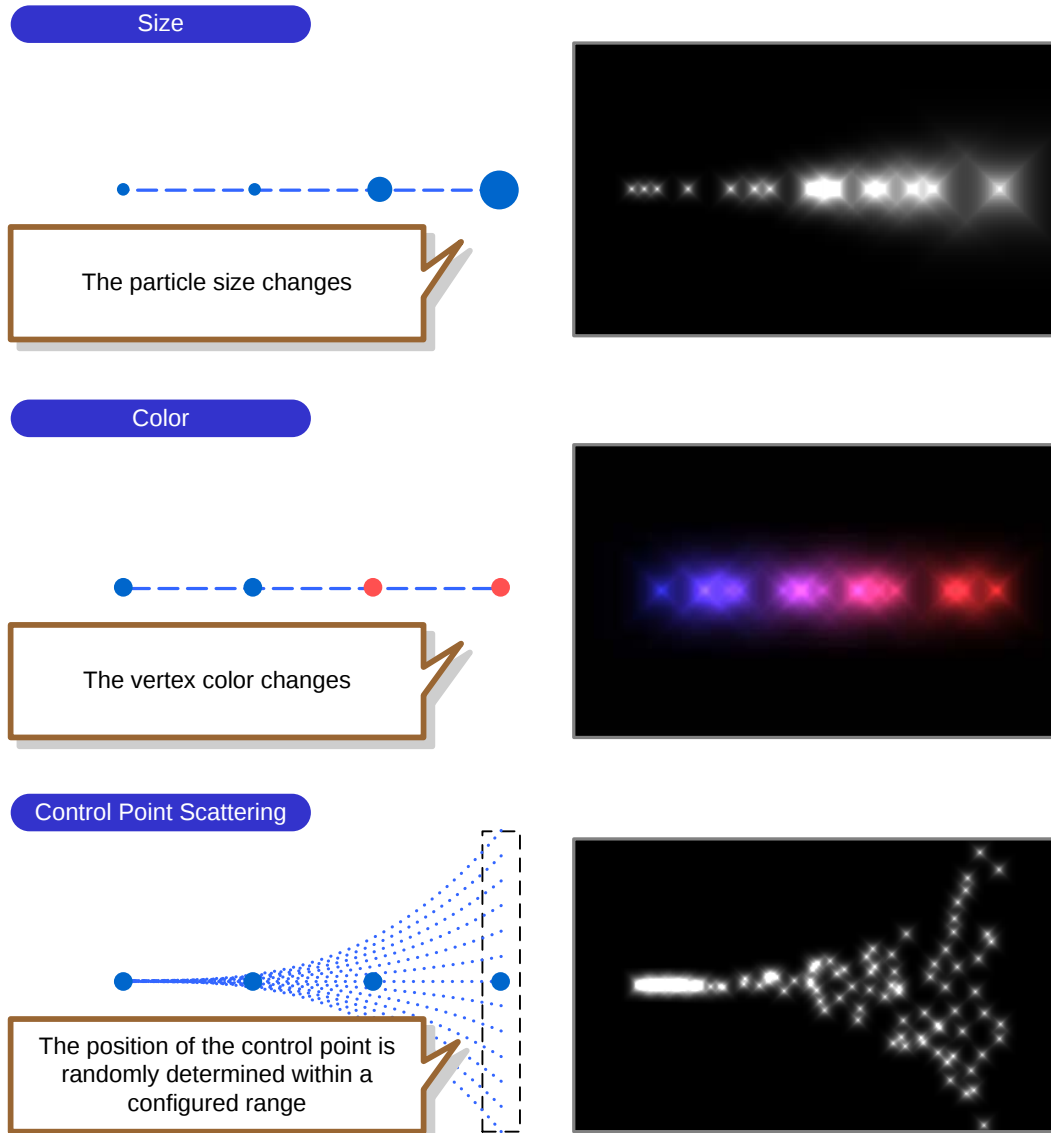
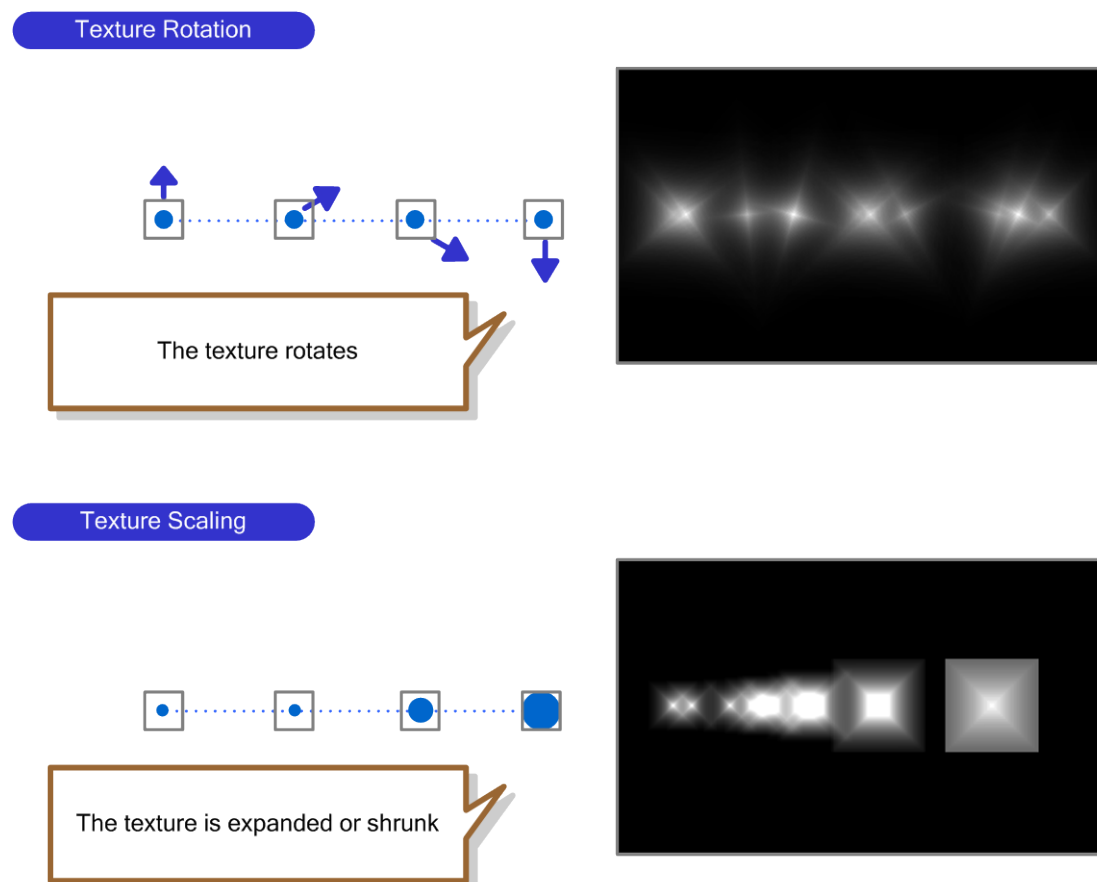
Figure 3-21 Control Point Settings

Figure 3-22 Individual Control Point Settings (Texture Coordinates)

Two types of texture coordinates are generated for the polygons created by the particle system: texture coordinate 0 and texture coordinate 2. Texture scaling is only valid for texture coordinate 2. For all other detailed texture settings, see Chapter 5 Textures.

Table 3-11 Particle Texture Coordinates

Item	Texture Coordinate 0	Texture Coordinate 2
Base coordinates		
Rotation	Valid	Valid
Scale	Not valid	Valid

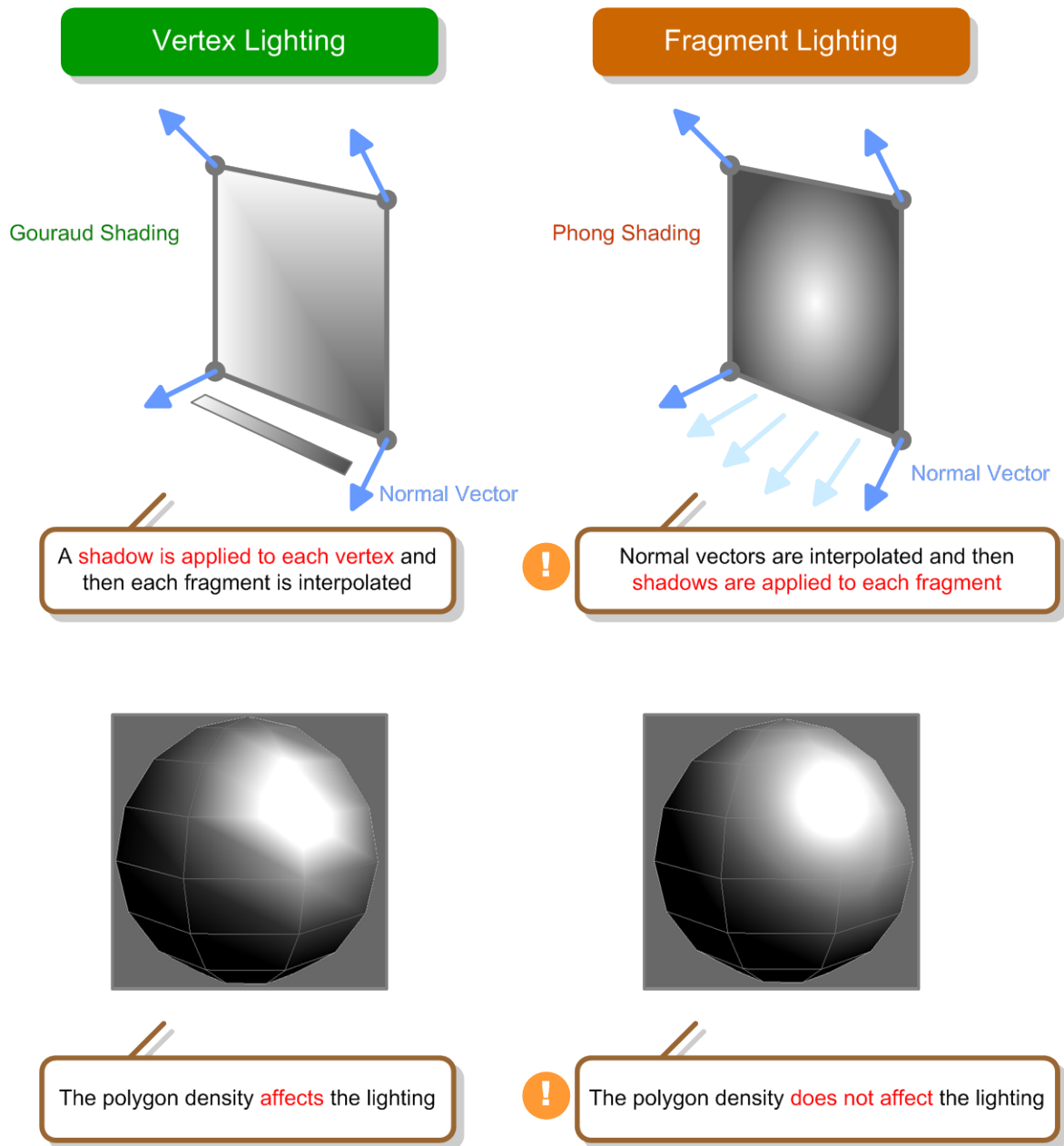
4 Fragment Lighting

4.1 What Is Fragment Lighting?

Fragment lighting is a process that calculates lighting one fragment at a time. Fragment lighting (Phong shading) can create more natural shadows than vertex lighting (Gouraud shading), which is calculated one vertex at a time.

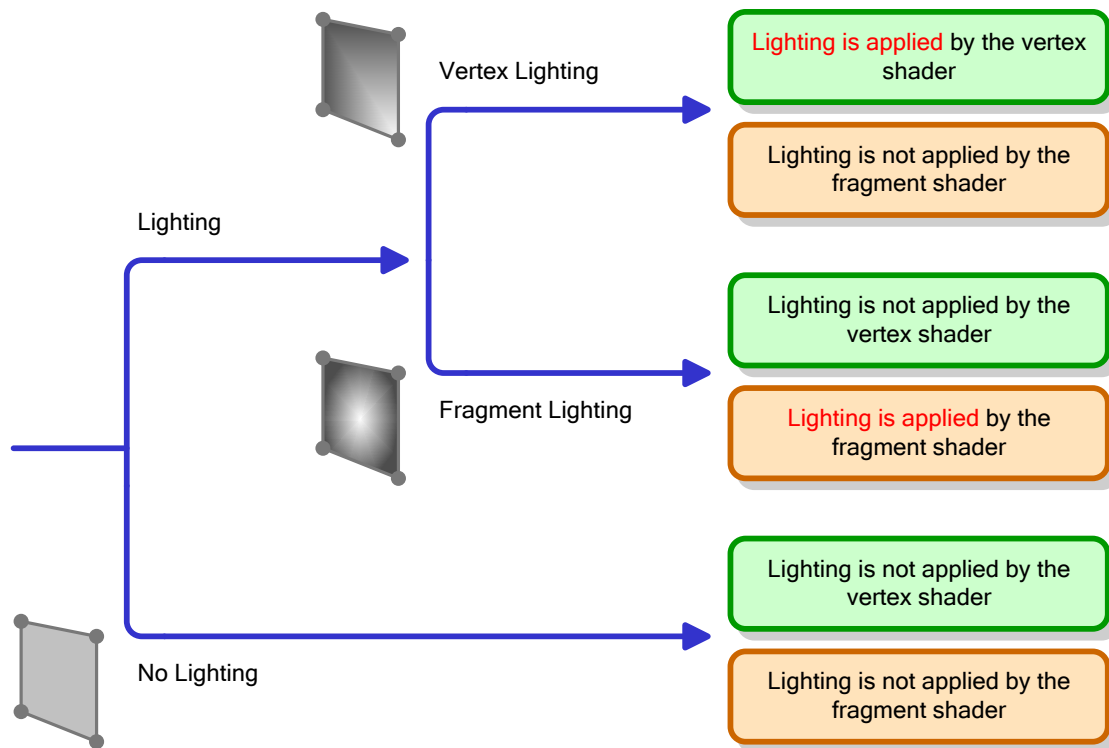
CTR fragment lighting can depict ordinary Phong shading as well as more complicated lighting.

Figure 4-1 Vertex Lighting and Fragment Lighting



Another possible method of lighting a polygon is vertex lighting, which is calculated within the vertex shader. You can also render without lighting.

Figure 4-2 Choice of Lighting Methods

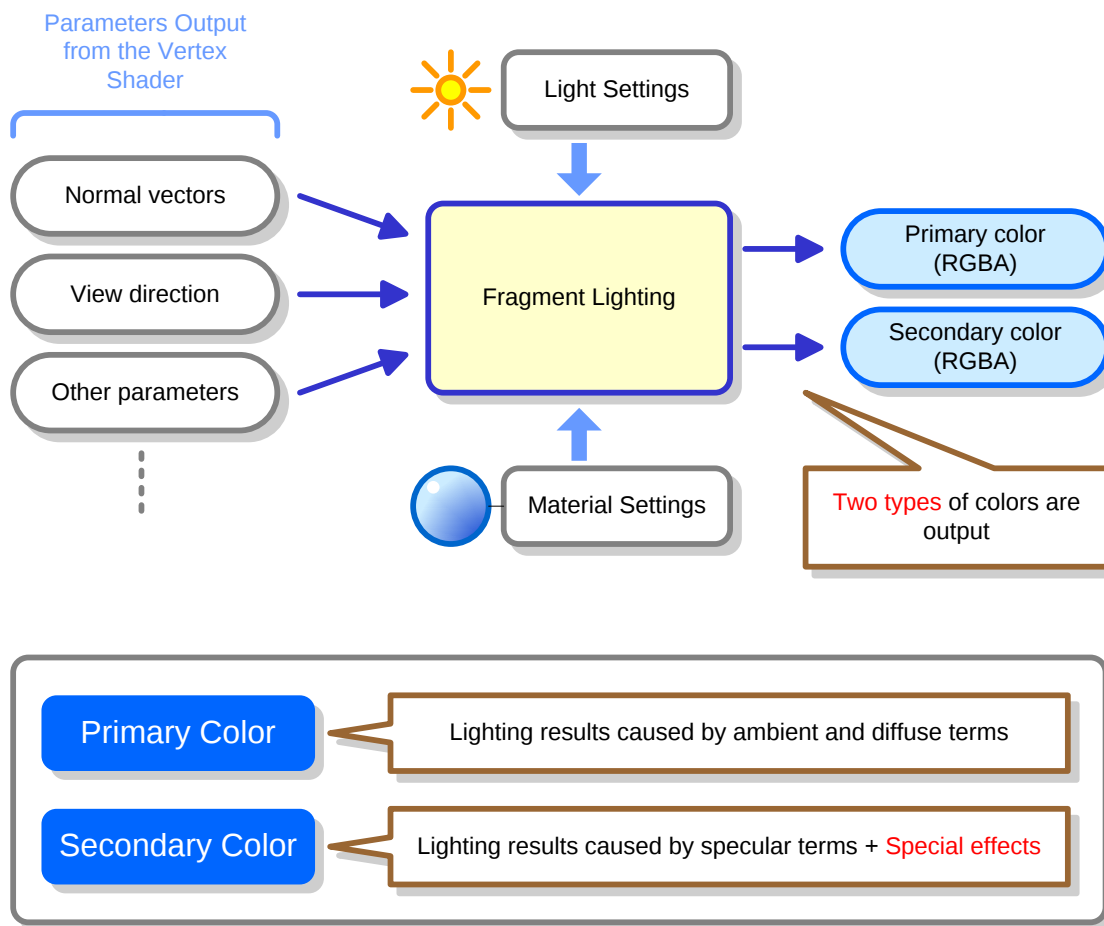


Note: You can use vertex lighting and fragment lighting simultaneously.

4.2 Fragment Lighting Overview

Fragment data output by the vertex shader is used as input to calculate fragment lighting from light and material settings. Two types of colors—primary and secondary—are output as the result of fragment lighting.

Figure 4-3 Fragment Lighting Overview

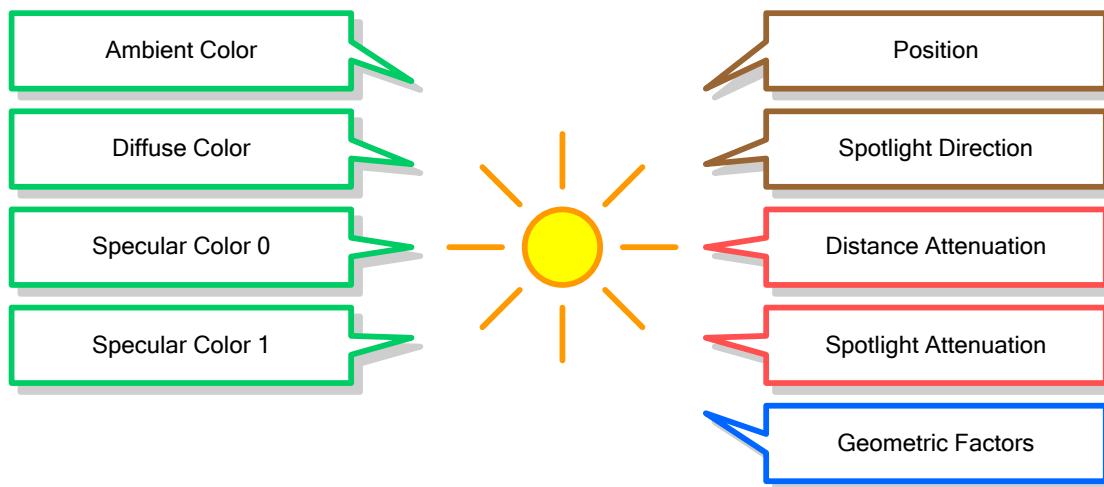


Note: The specular representation (secondary color) can be used for various settings. The secondary color is therefore an important element for representing materials on the CTR system.

4.3 Light Settings

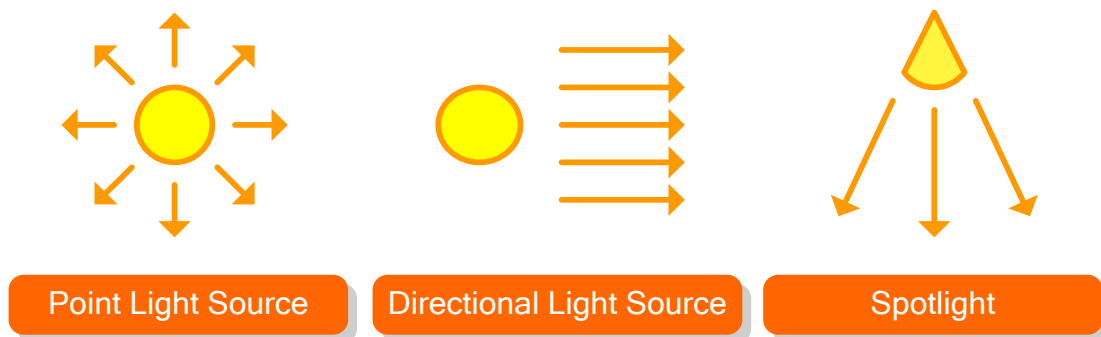
You can configure the following items for each light.

Figure 4-4 Light Settings



The following three types of light sources can be represented by light settings.

Figure 4-5 Types of Light Sources



4.3.1 Number of Lights

You can set up to eight lights for each material.

Note: The cost to process fragment lighting increases with the number of lights used.

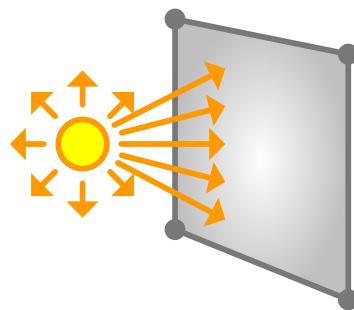
4.3.2 Position

You can set the position of a light. You can place a light close to an object to depict a point light source. You can place a light infinitely far away to depict a directional light source like the sun.

Figure 4-6 Light Source Depictions Due to Light Positions

Point Light Source

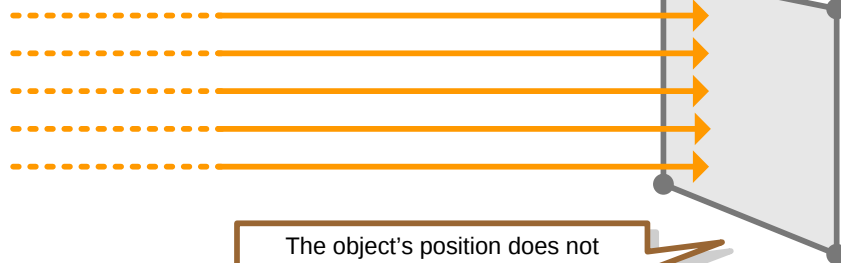
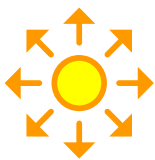
Place a light close to an object



The object's position changes the light angle

Directional Light Source

Place a light infinitely far away

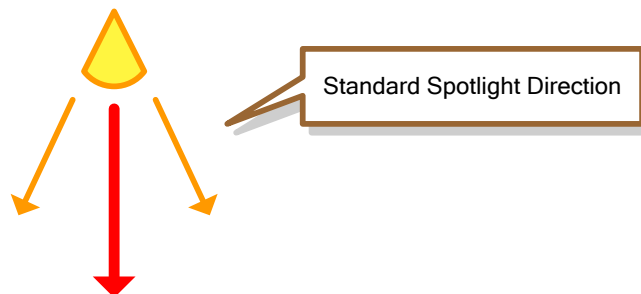


The object's position does not change the light angle

4.3.3 Spotlight Direction

You can configure a light's standard spotlight direction.

Figure 4-7 Spotlight Direction



4.3.4 Light Colors

You can set the following four types of light colors.

Table 4-1 Light Colors

Light Colors	Description
Ambient color	Represents environmental light.
Diffuse color	Represents diffuse light.
Specular color 0	Represents specular light 0.
Specular color 1	Represents specular light 1.

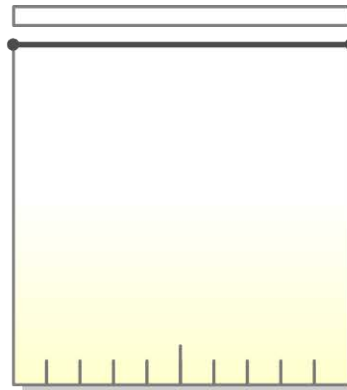
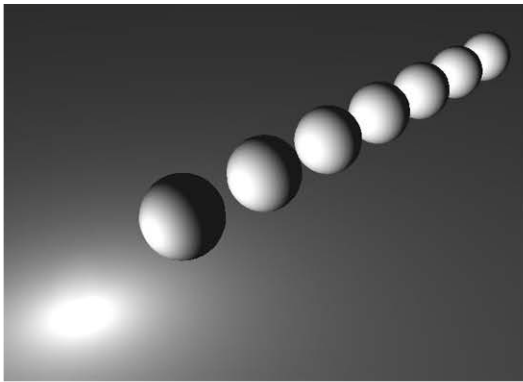
4.3.5 Distance Attenuation

This configures how a light is attenuated by its distance to an object's surface.

Distance attenuation settings use lookup tables. For more information on lookup tables, see section 4.6 Effects That Use Lookup Tables.

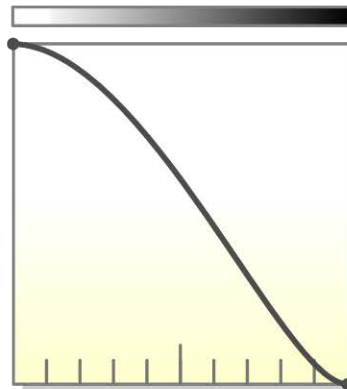
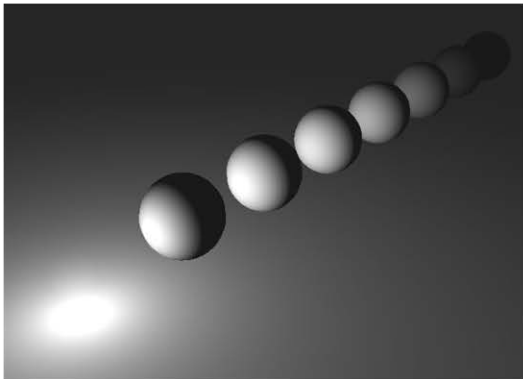
Figure 4-8 Distance Attenuation

Example 1 No distance attenuation



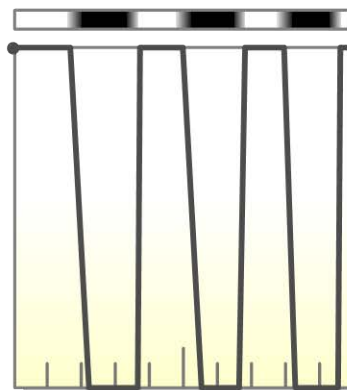
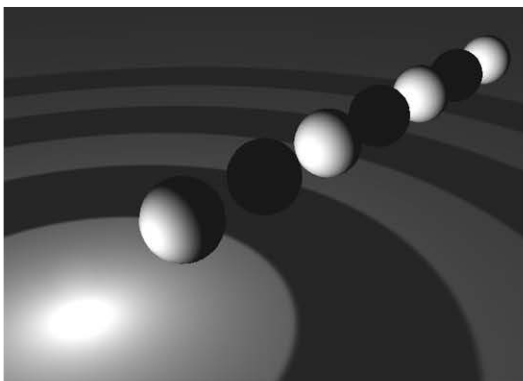
Lookup Table

Example 2 Normal distance attenuation



Lookup Table

Example 3 Banded distance attenuation



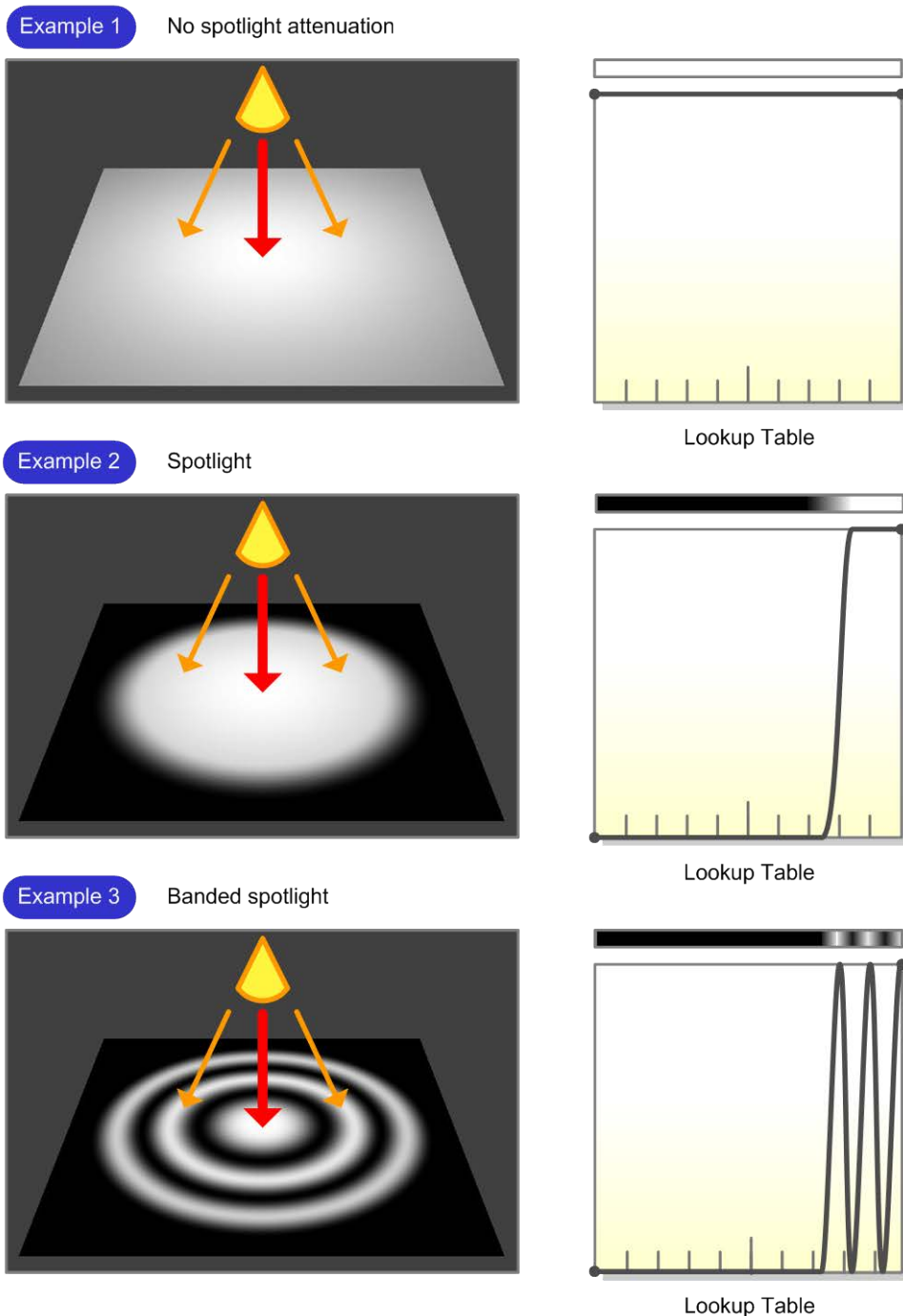
Lookup Table

4.3.6 Spotlight Attenuation

This configures how light attenuates with respect to the spotlight direction.

Spotlight attenuation settings use lookup tables. For more information on lookup tables, see section 4.6 Effects That Use Lookup Tables.

Figure 4-9 Spotlight Attenuation



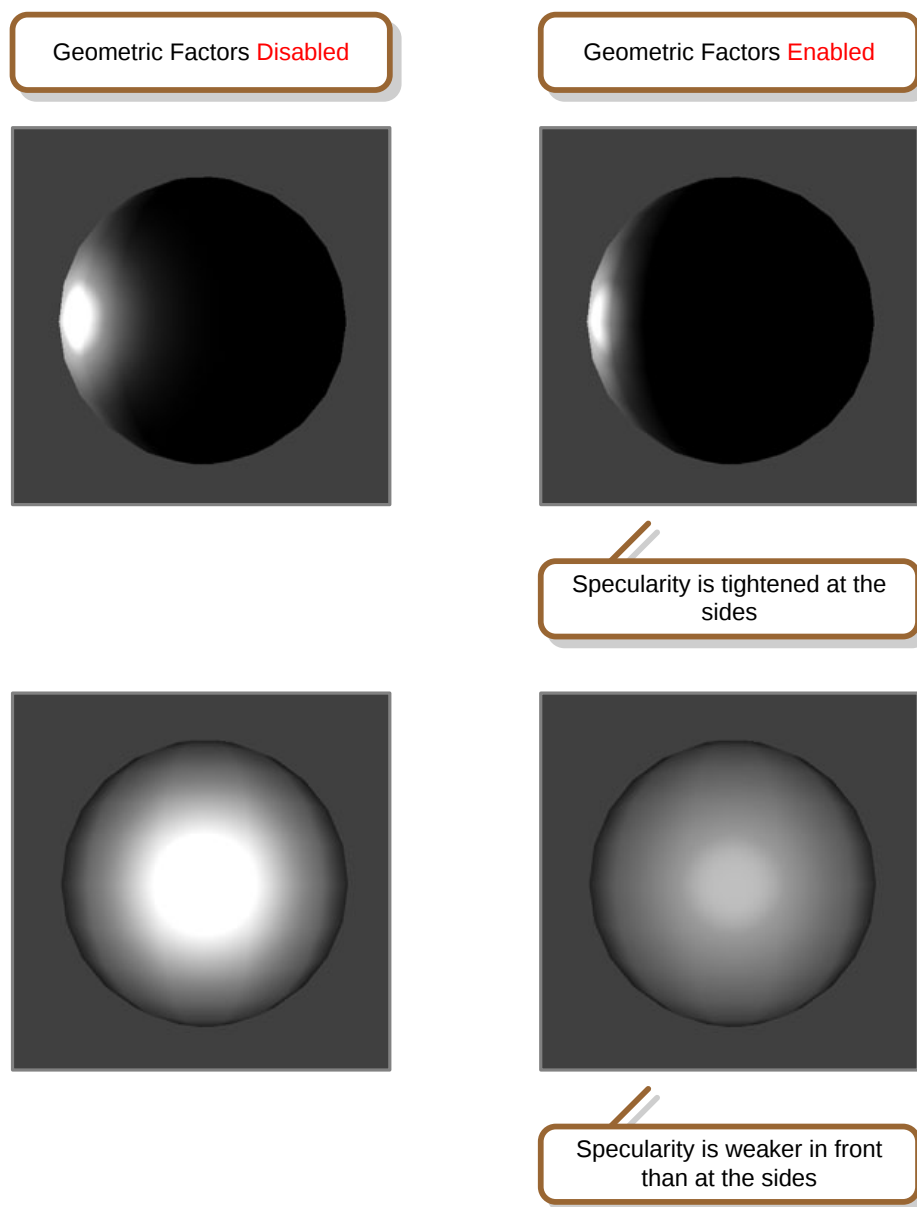
4.3.7 Geometric Factors

Geometric factors are settings used to adjust specular appearance.

They can be either enabled or disabled. When enabled, the specularity is tightened at the lateral (side) faces and the specularity of the front face becomes weaker than the lateral faces (the front and lateral faces are taken with respect to the view direction).

The use of geometric factors enhances metallic representations.

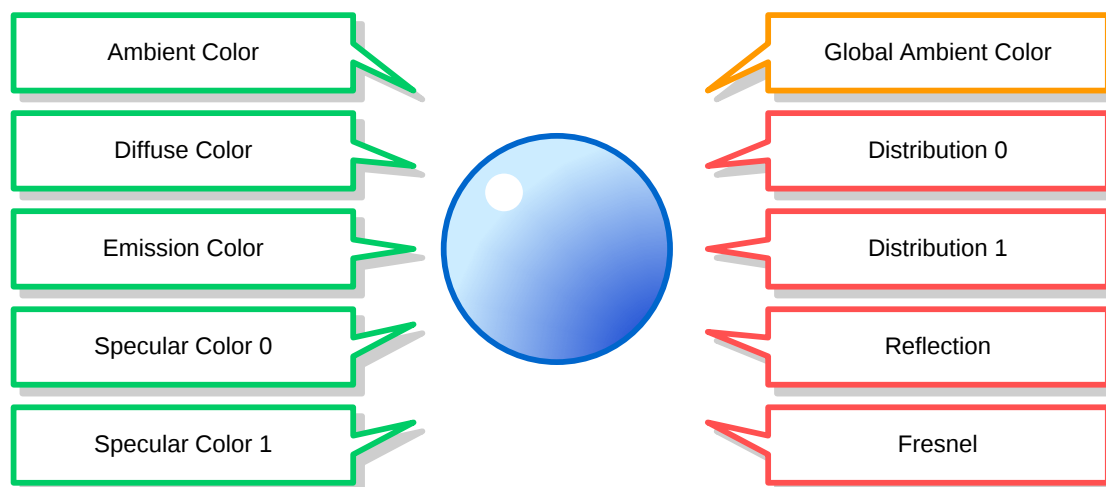
Figure 4-10 Geometric Factors



4.4 Material Settings

You can configure the following items for each material.

Figure 4-11 Material Settings



4.4.1 Material Colors

You can set the following five types of material color.

Table 4-2 Material Colors

Material Colors	Description
Ambient color	Represents the ambient color.
Diffuse color	Represents the diffuse color.
Emission color	Represents the emission color.
Specular color 0	Represents specular color 0.
Specular color 1	Represents specular color 1.

4.4.2 Global Ambient Color

The global ambient color is used as the global light in fragment lighting calculations. Although this is actually a light environment setting rather than a material setting, you can configure it for each material.

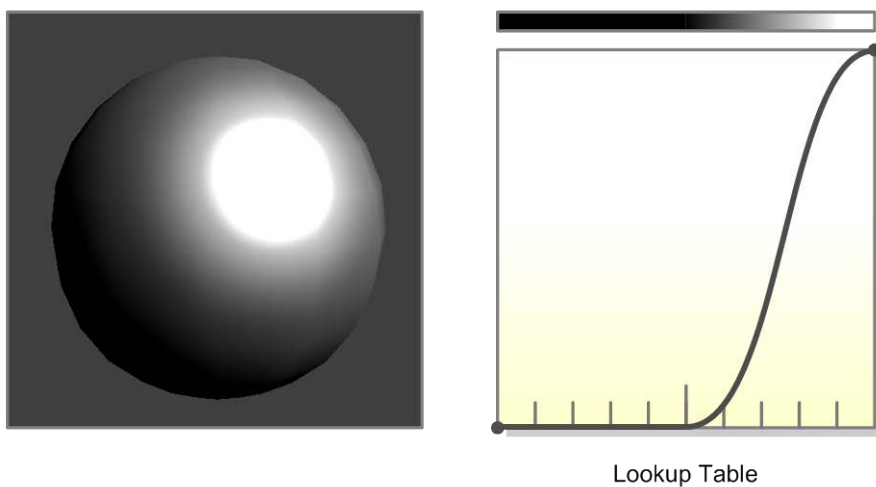
4.4.3 Distribution (Specular Shape)

Distribution settings are used to control the specular shape.

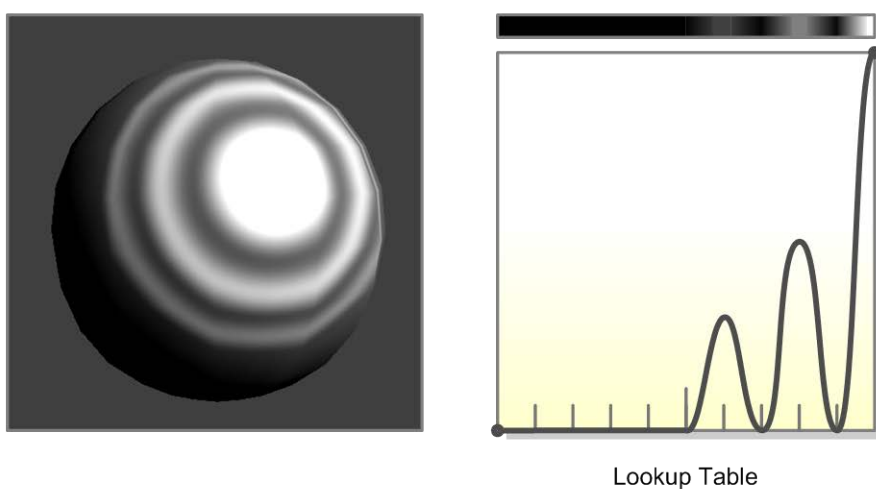
Distribution settings use lookup tables. For more information on lookup tables, see section 4.6 Effects That Use Lookup Tables.

Figure 4-12 Distribution

Example 1 Ordinary Specular Representation



Example 2 Banded Specular Representation



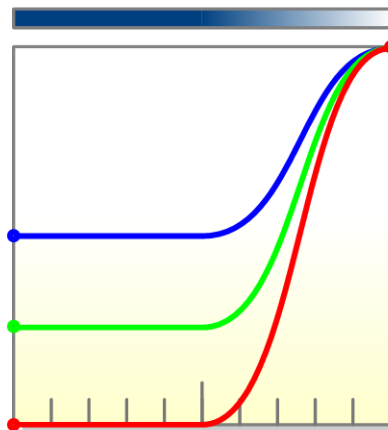
4.4.4 Reflection (Reflected Colors)

Reflection settings are used to control reflected colors.

Reflection settings use lookup tables. For more information on lookup tables, see section 4.6 Effects That Use Lookup Tables.

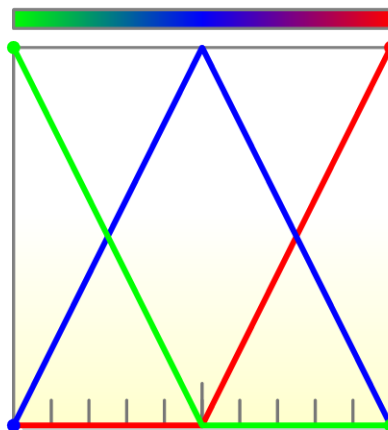
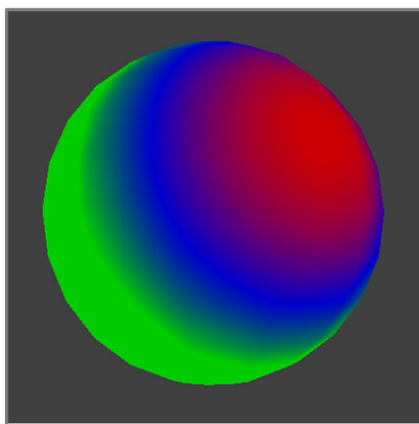
Figure 4-13 Reflection

Example 1 Colored Reflection



Lookup Table

Example 2 Reflection Whose Colors Change By Angle



Lookup Table

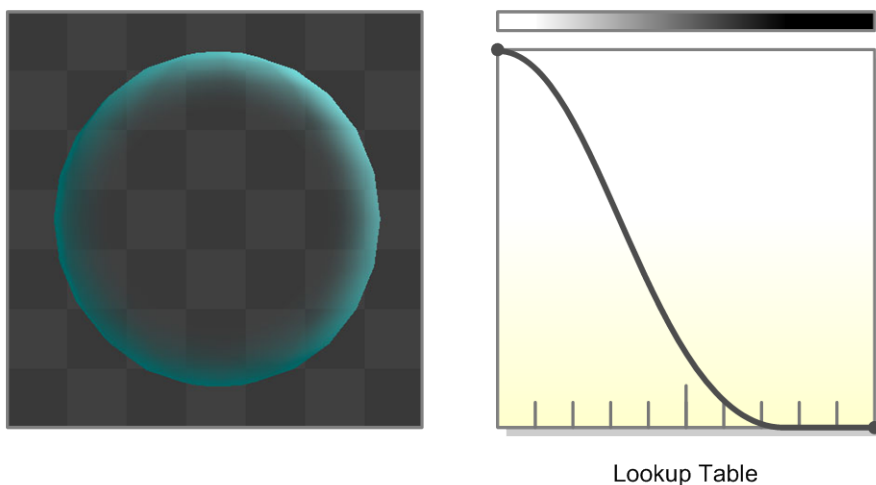
4.4.5 Fresnel (Transparency)

Fresnel settings are used to control material transparency (alpha).

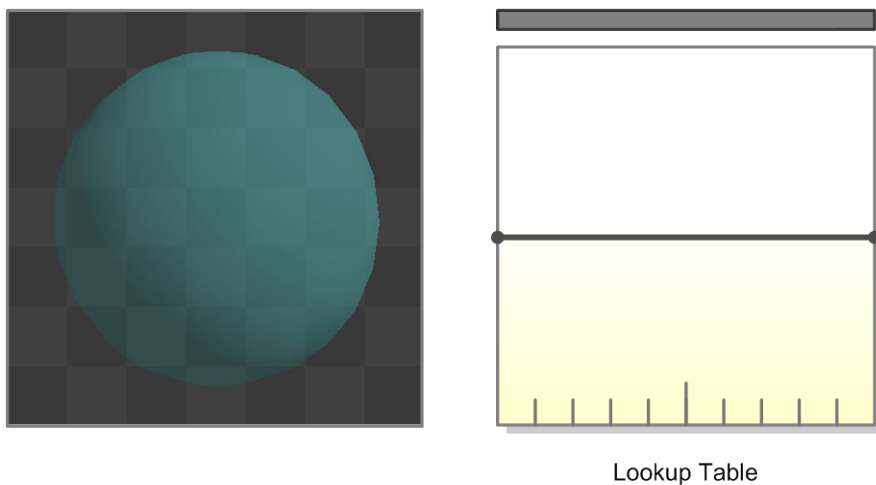
Fresnel settings use lookup tables. For more information on lookup tables, see section 4.6 Effects That Use Lookup Tables.

Figure 4-14 Fresnel

Example 1 Representation in Which the Transparency Changes by Angle



Example 2 Representation With the Same Overall Transparency



Note: This item is named "Fresnel" but it is only used to control transparency. It is not restricted to representing general Fresnel reflections.

4.5 Lighting Formulas

Fragment lighting calculations are split between the primary color, which represents the effect of ambient and diffuse terms, and the secondary color, which represents the effect of specular terms.

Figures in the next sections use the following icons to simplify lighting formula descriptions.

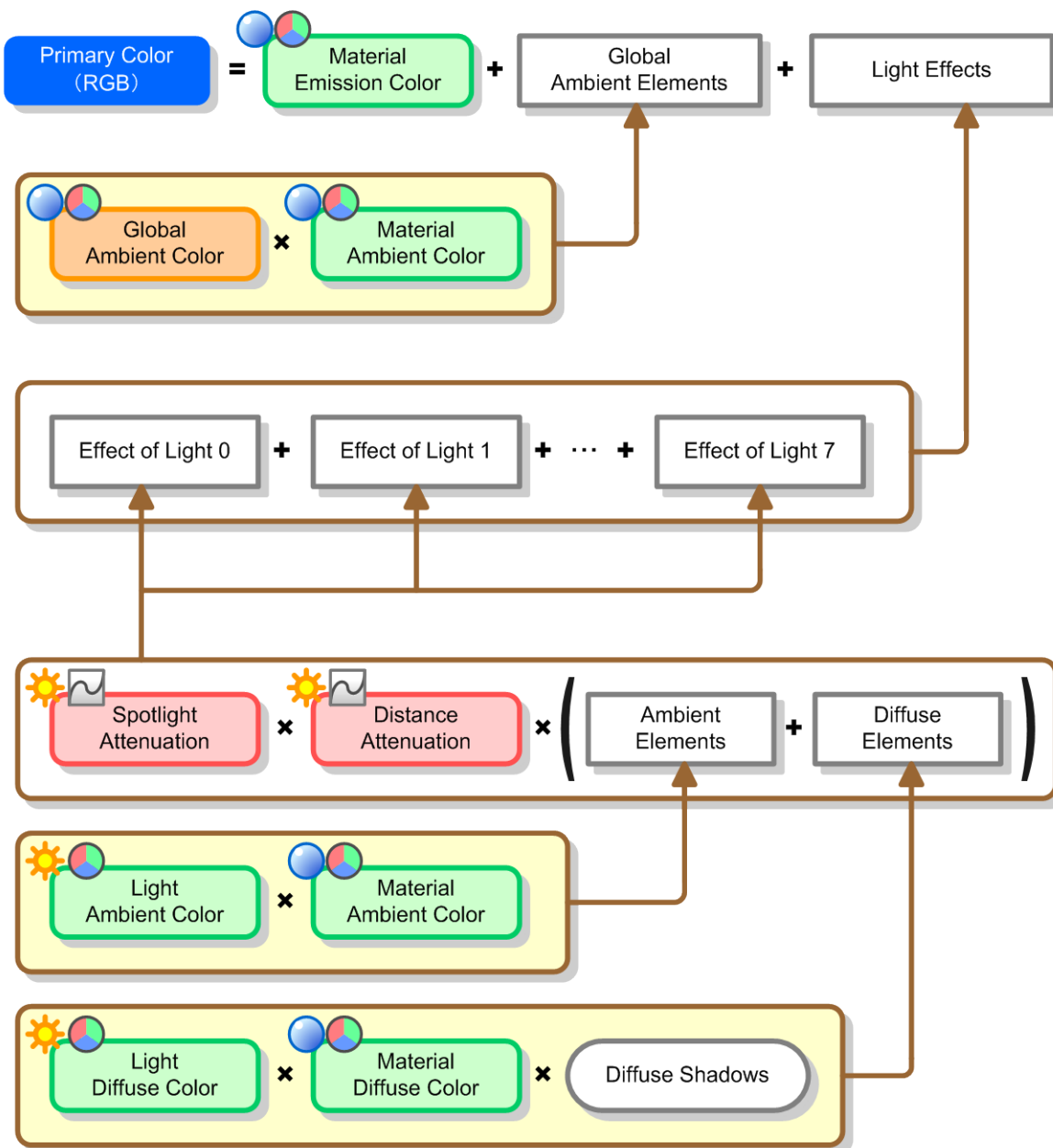
Table 4-3 Icons Used in Lighting Formulas

Icon	Description
	Indicates an item that is set for each light.
	Indicates an item that is set for each material.
	Indicates an item that sets the (RGB) color.
	Indicates an item that sets a lookup table.

4.5.1 Primary Color

The primary color uses the following formula.

Figure 4-15 Primary Color Formula

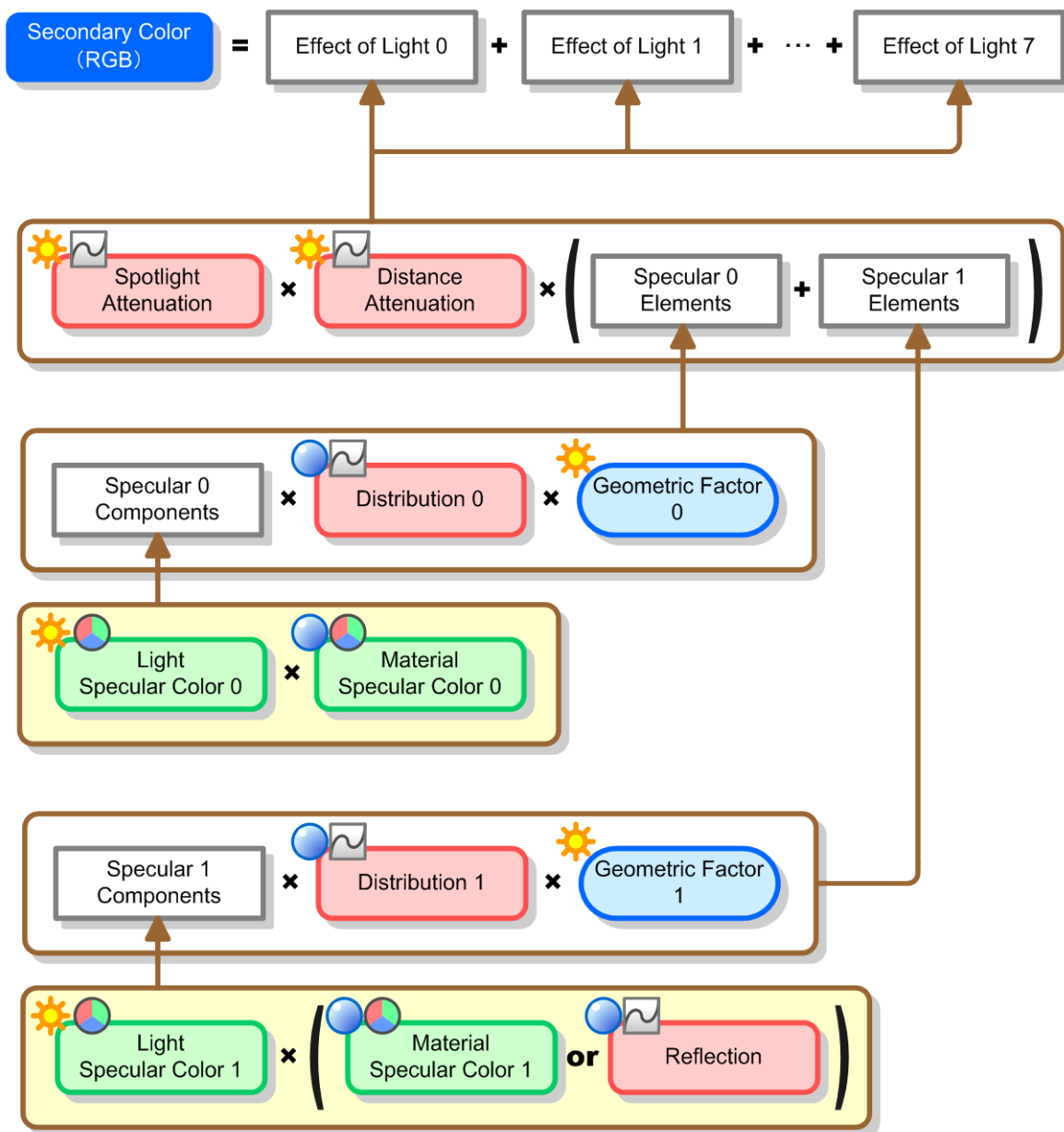


Note: Because of hardware restrictions, the material emission color and the global ambient elements are only valid when light 0 is applied. To use only the material emission color and the global ambient elements, apply light 0 with each color set equal to 0 (black).

4.5.2 Secondary Color

The formula for the secondary color comprises only the specular effects of each light. The secondary color is always 0 (black) when no lights have been set. The secondary color uses the following formula.

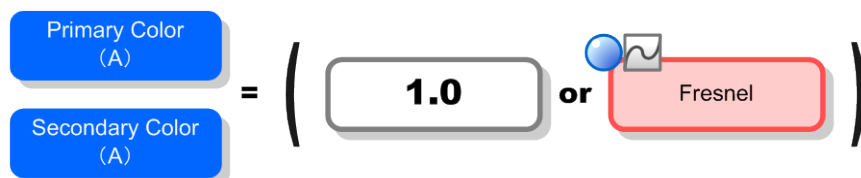
Figure 4-16 Secondary Color Formula



4.5.3 Alpha

The alpha component of the primary and secondary colors is determined by the Fresnel value set for a material. Fragment lighting always outputs an alpha value of 1 (opaque) when a Fresnel value has not been set.

Figure 4-17 Alpha Formula



Note: Only the enabled light with the largest number is applied for alpha representations that depend on the light position (alpha lighting).

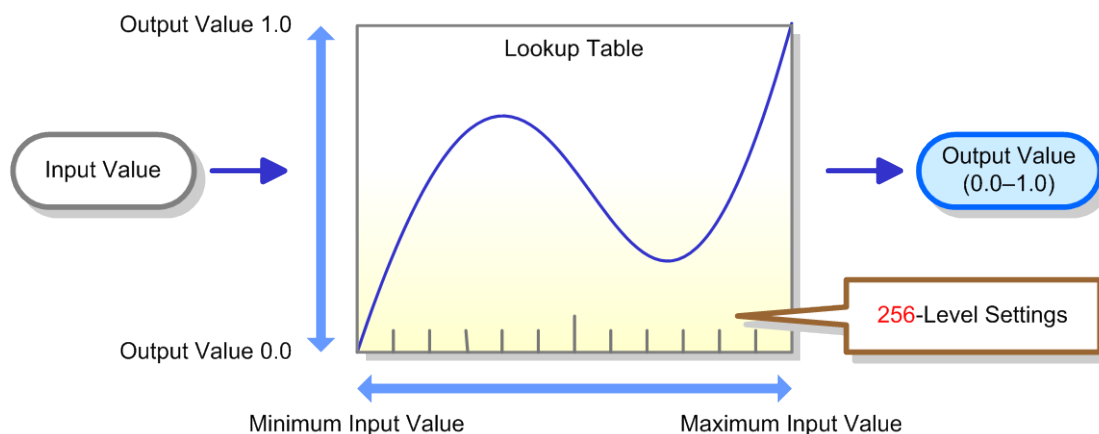
4.6 Effects That Use Lookup Tables

Fragment lighting uses lookup tables for the following settings.

Table 4-4 Settings That Use Lookup Tables

Setting	Units	Main Use
Distance attenuation	Lights	Distance attenuation
Spotlight attenuation	Lights	Spotlights
Distribution	Materials	Specular shape
Reflection	Materials	Reflected color
Fresnel	Materials	Transparency

The lookup tables represent curves with an output value set for each input value. Lighting is calculated using output values that are set in advance for 256 input values.

Figure 4-18 Lookup Table Structure

4.6.1 Lookup Table Input Values

Lookup table input values are selected from the following six types.

Table 4-5 Lookup Table Input Values

Input Value	Description	Main Use
LN	Angle between the light vector and normal vector	Diffuse
NV	Angle between the normal vector and view vector	Fresnel
NH	Angle between the normal vector and half vector	Specular
VH	Angle between the view vector and half vector	
SP	Angle between the inverse light vector and spotlight direction	Spotlight attenuation
CP	Angle between the projection of the half vector onto the tangent plane and the tangent vector	Anisotropic reflections

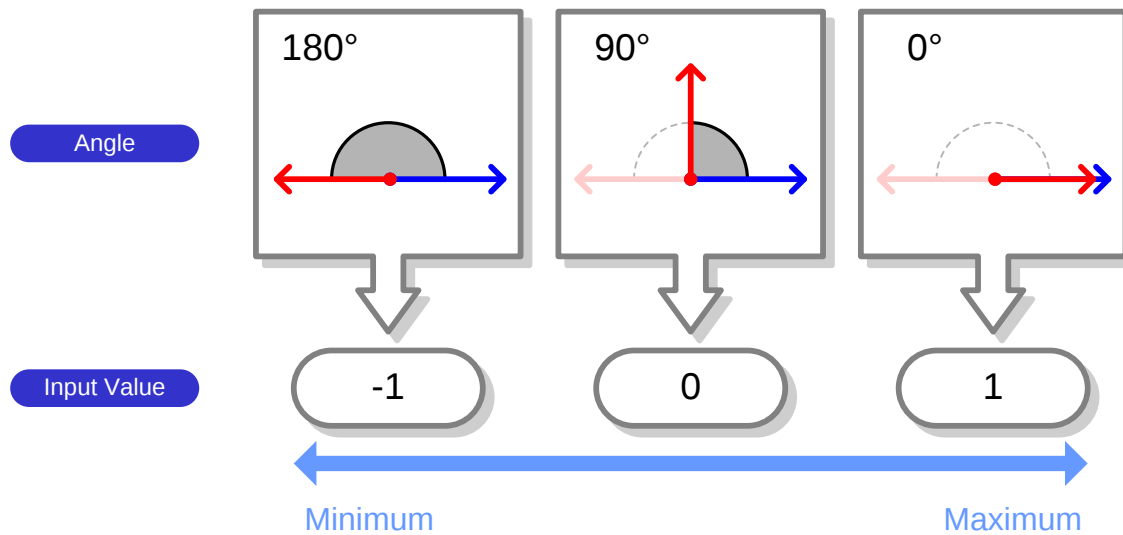
There are the following restrictions on settings that use lookup tables and on the combinations of lookup table input values.

Table 4-6 Settings Using Lookup Tables and Input Value Restrictions

Setting	Input Value					
	LN	NV	NH	VH	SP	CP
Distance Attenuation	The distance from the light to the object's surface is always used.					
Spotlight Attenuation	✓	✓	✓	✓	✓	✓
Distribution	✓	✓	✓	✓	✓	✓
Reflectance	✓	✓	✓	✓	-	-
Fresnel	✓	✓	✓	✓	-	-

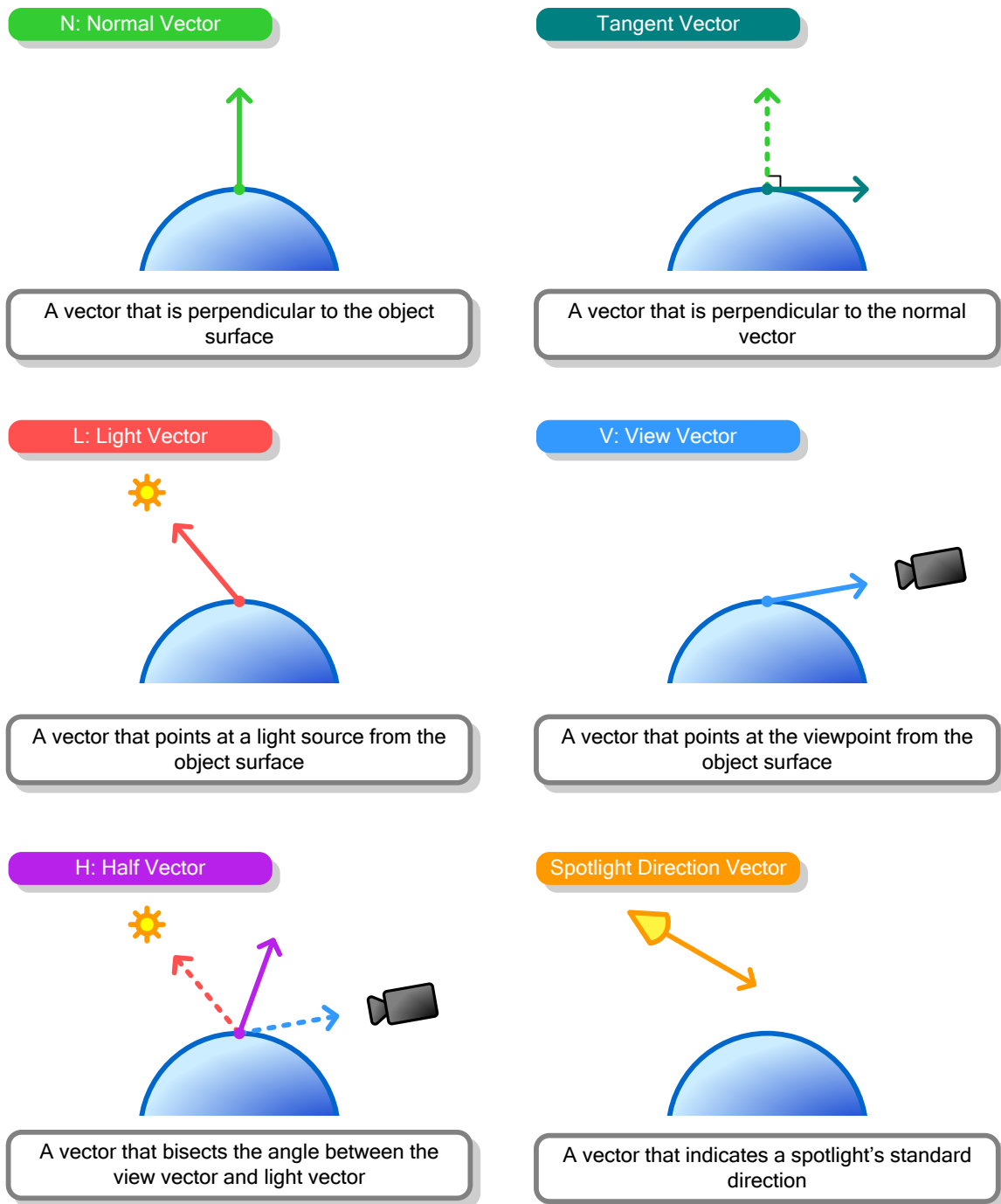
Each input value is determined from the angle between two vectors. The input has a minimum value of -1 for an angle of 180 degrees and a maximum value of 1 for an angle of 0 degrees.

Figure 4-19 Relationship Between the Input Value and the Angle Created by Two Vectors



Each input value is determined from the following six vector combinations.

Figure 4-20 Input Values and Base Vectors

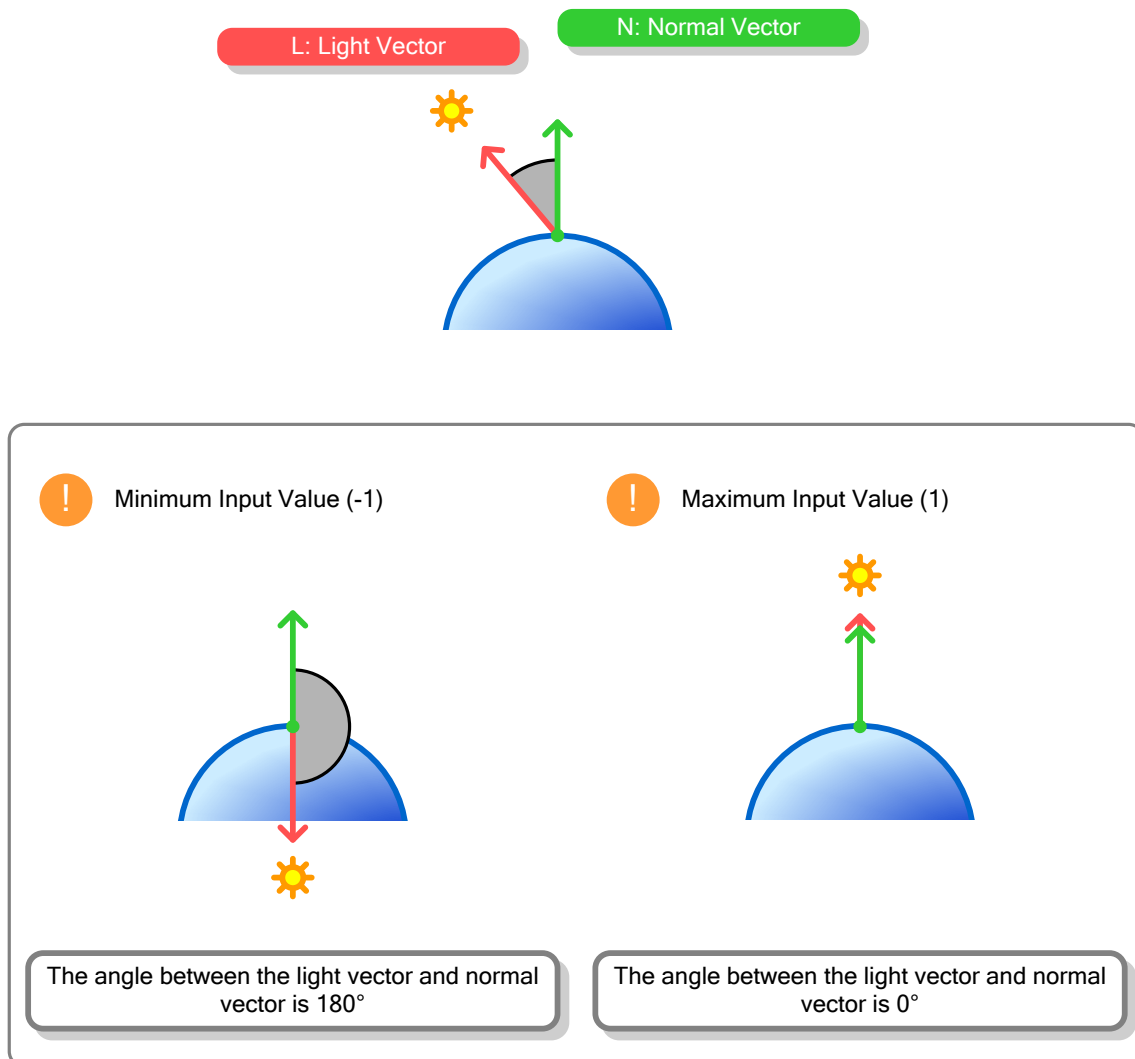


Note: The light vector and view vector indicate the direction from the surface of the object to each element.

4.6.1.1 LN Input Value

LN input values are determined by the angle between the light vector and normal vector. LN input values are mainly used for diffuse representations.

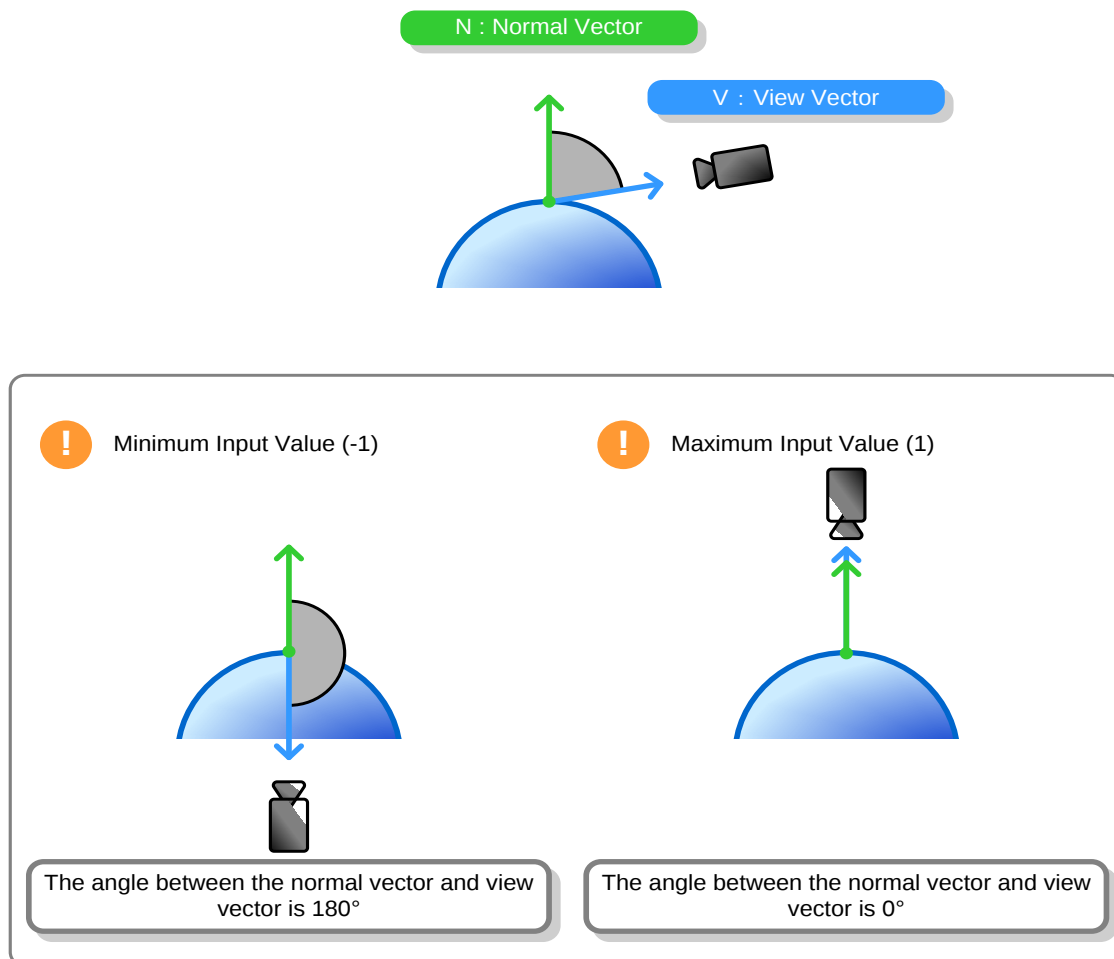
Figure 4-21 LN Input Value



4.6.1.2 NV Input Value

NV input values are determined by the angle between the normal vector and the view vector. NV input values are mainly used to represent glass and water surfaces, for which reflectivity (Fresnel reflection) changes with the viewing angle.

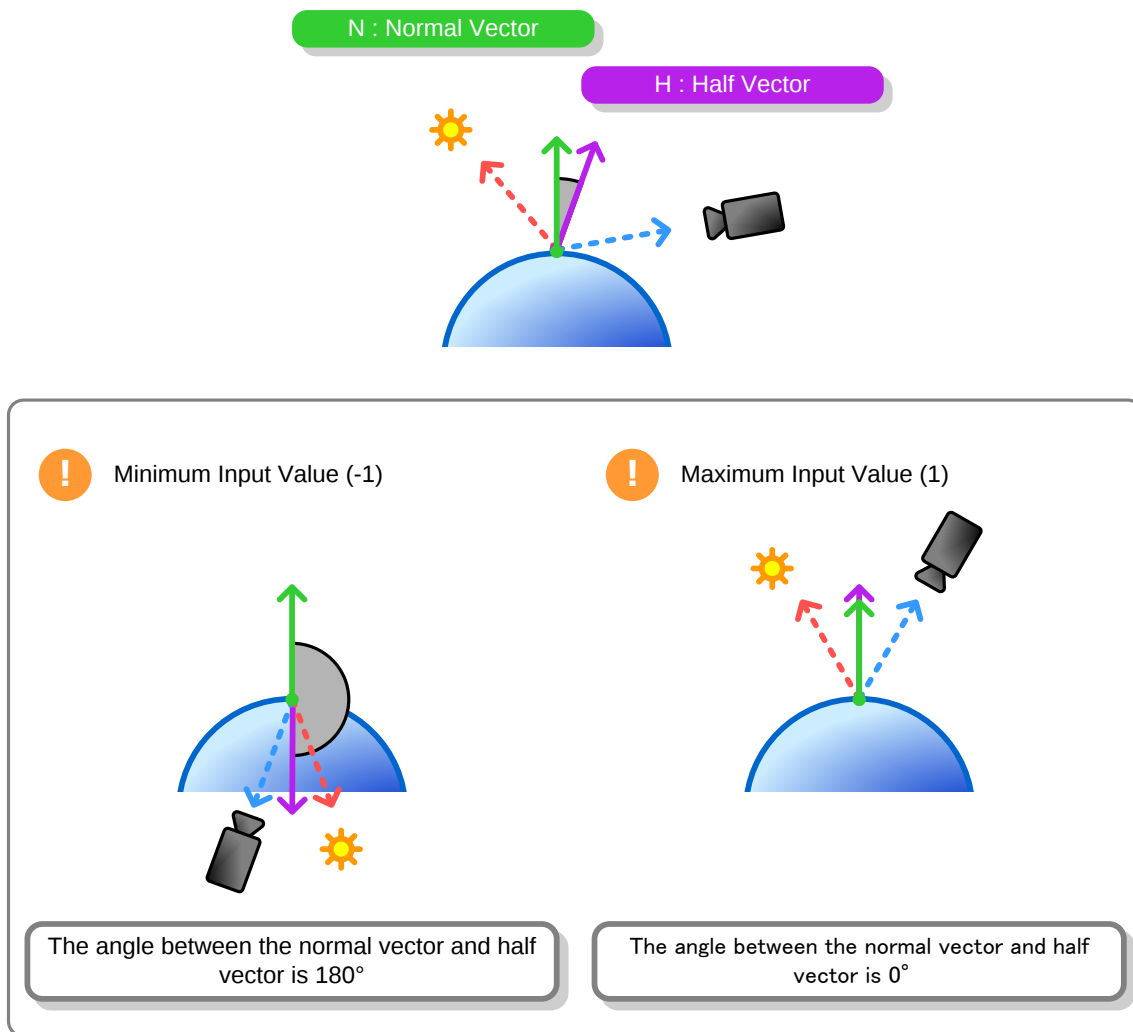
Figure 4-22 NV Input Value



4.6.1.3 NH Input Value

NH input values are determined by the angle between the normal vector and the half vector. NH input values are mainly used for specular representations.

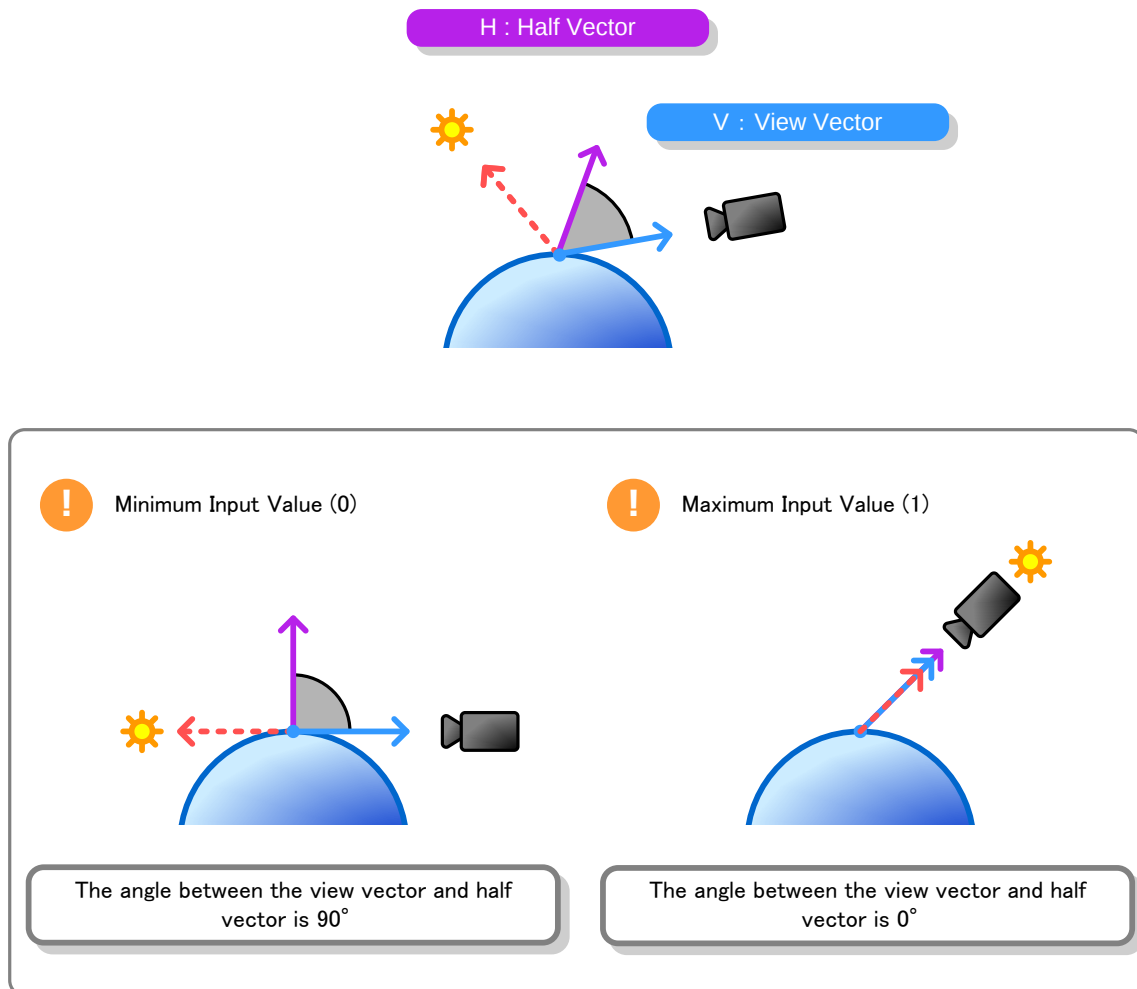
Figure 4-23 NH Input Value



4.6.1.4 VH Input Value

VH input values are determined by the angle between the view vector and the half vector.

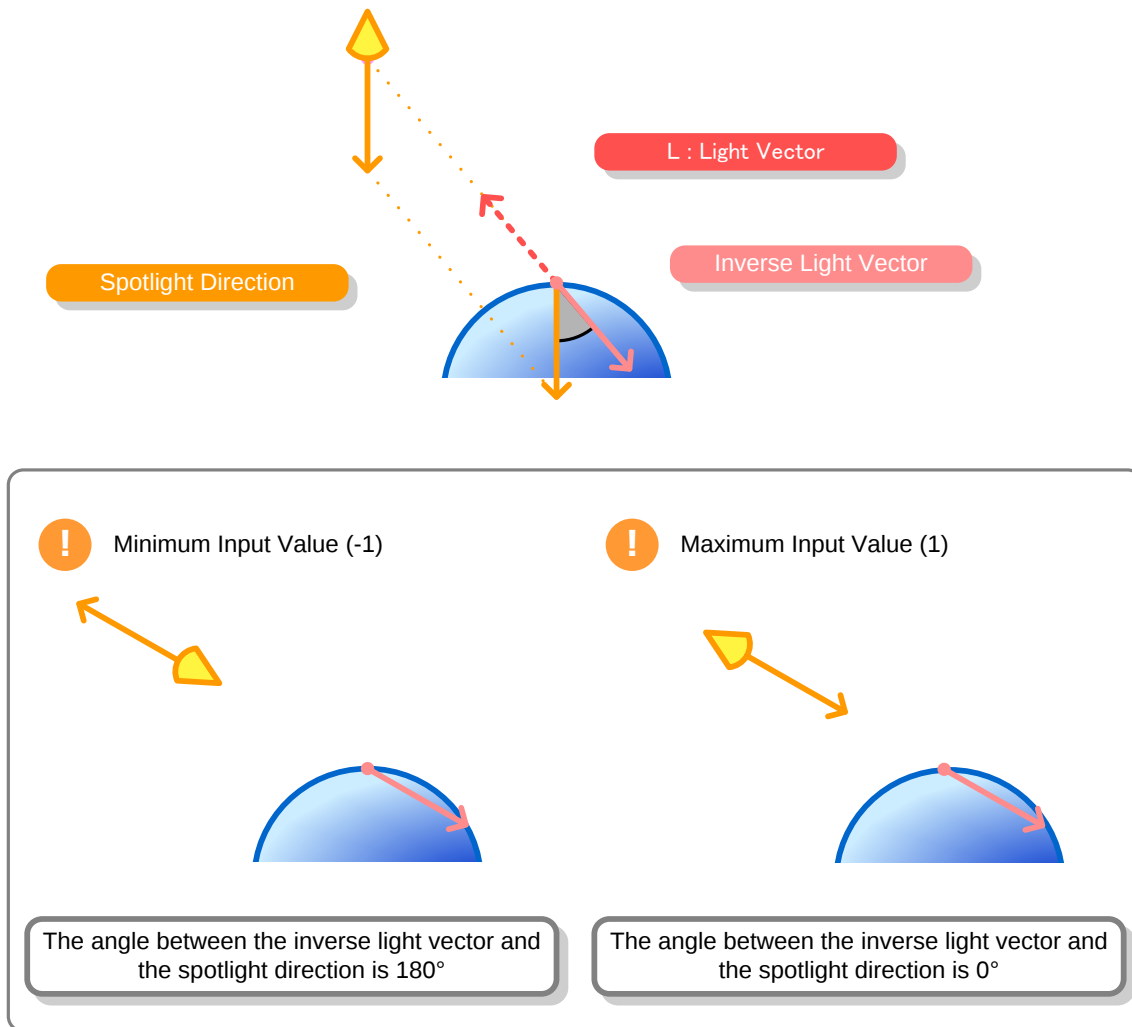
Figure 4-24 VH Input Value



4.6.1.5 SP Input Value

SP input values are determined by the angle between the inverse light vector and the spotlight direction. SP input values are mainly used to depict spotlights.

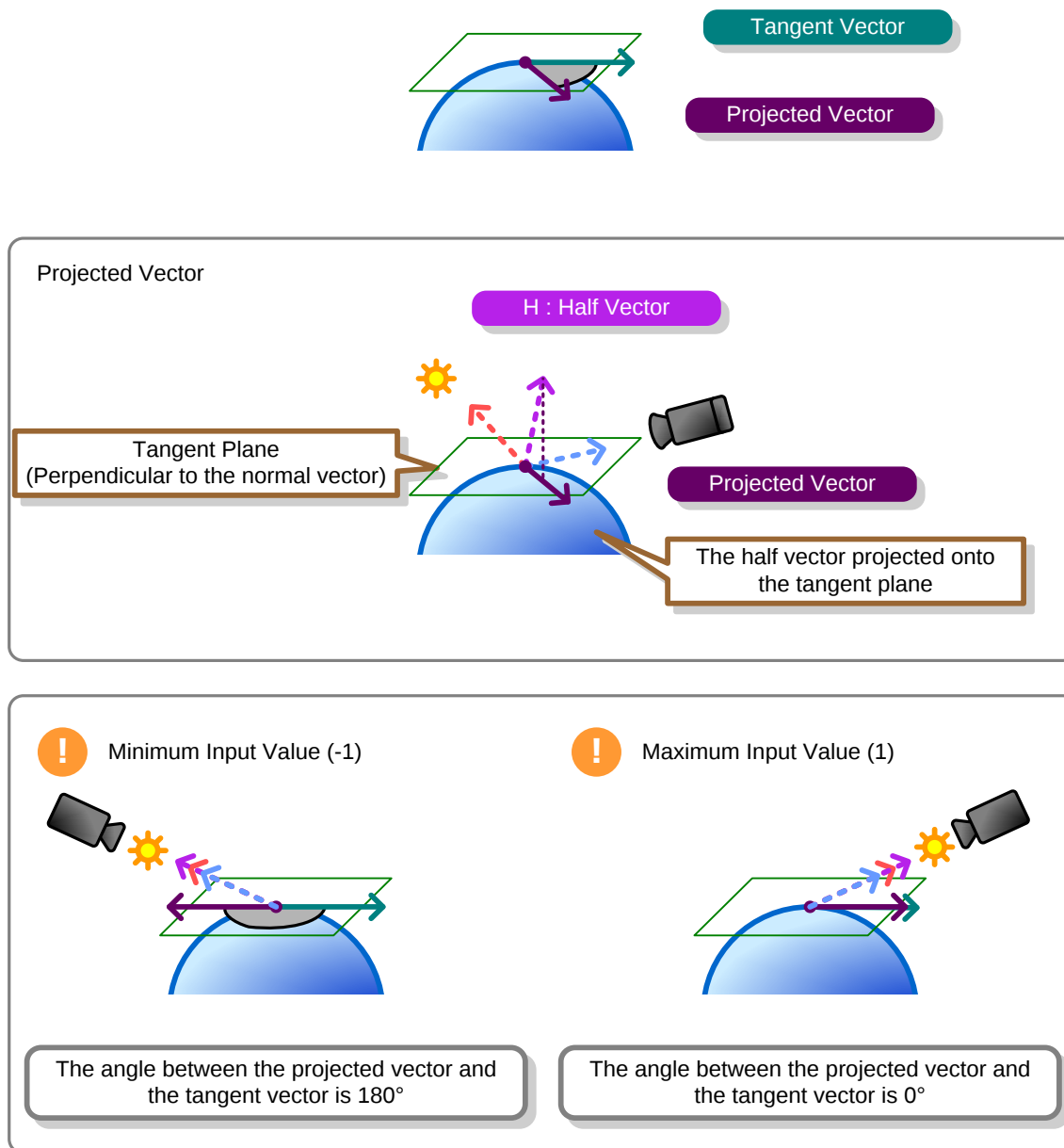
Figure 4-25 SP Input Value



4.6.1.6 CP Input Value

CP input values are determined by the angle between the projection of the half vector onto the tangent plane (parallel projection) and the tangent vector. CP input values are mainly used to represent anisotropic reflections.

Figure 4-26 CP Input Value



Note: You must set layer configuration 7 when using CP input values. For more information on layer configurations, see section 4.7 Layer Configurations.

4.7 Layer Configurations

Usable lookup table combinations are fixed for fragment lighting calculations. These combinations are called *layer configurations*.

Lighting is not affected by items for which a lookup table is not applied (they are each handled as a value of 1 in the lighting formulas). The layer configuration that you use changes the cost to process lighting.

Table 4-7 Layer Configurations

Layer Configuration	Processing Load	Lookup Tables					
		Distance Attenuation	Spotlight Attenuation	Distribution 0	Distribution 1	Reflection	Fresnel
0	x1	YES	YES	YES	-	Grayscale	-
1	x1	YES	YES	-	-	Grayscale	YES
2	x1	YES	-	YES	YES	Grayscale	-
3	x1	YES	-	YES	YES	-	YES
4	x3	YES	YES	YES	YES	YES	-
5	x3	YES	YES	YES	-	YES	YES
6	x3	YES	YES	YES	YES	Grayscale	YES
7	x4	-	YES	YES	YES	YES	YES

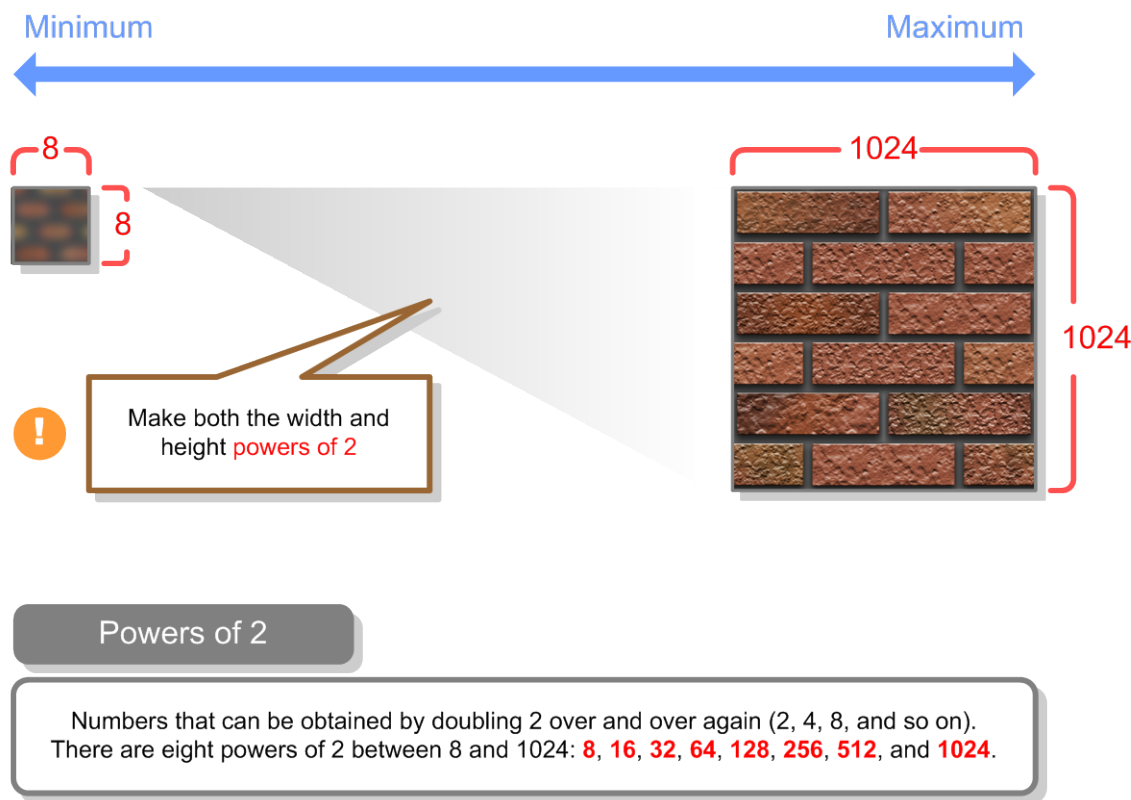
Note: Where indicated, the reflection lookup table is used as grayscale values (one component) rather than colors (three RGB components).

5 Textures

5.1 Texture Size

The CTR system can use textures with 8-1024 texels. The width and height do not need to be the same; rectangular textures are supported. However, you must use a width and height that are each a power of 2.

Figure 5-1 Texture Size



Note: The cost of processing a texture increases with its size.

Some texture formats have a minimum size of 16x16. For details, see each texture format's description.

5.2 Texture Formats

Texture formats with different uses and data sizes have been provided for CTR. You can choose a texture format based on its use and data size.

5.2.1 Types of Texture Formats

You can use the following 14 types of textures formats on the CTR system.

Table 5-1 Types of Texture Formats

Format Type	Texture Format	Internal Structure (Bits)	Bits/Texel	Transparency	Minimum Size
RGB formats	RGB565	R5 G6 B5	16	-	8 × 8
	RGB8	R8 G8 B8	24	-	8 × 8
RGBA formats	RGBA4	R4 G4 B4 A4	16	16 levels	8 × 8
	RGB5A1	R5 G5 B5 A1	16	2 levels	8 × 8
	RGBA8	R8 G8 B8 A8	32	256 levels	8 × 8
Alpha formats	A4	A4	4	16 levels	8 × 8
	A8	A8	8	256 levels	8 × 8
Luminance formats	L4	R4	4	-	8 × 8
	L8	R8	8	-	8 × 8
Luminance alpha formats	LA4	R4 A4	8	16 levels	8 × 8
	LA8	R8 A8	16	256 levels	8 × 8
ETC compressed formats	ETC	See note	4	-	16 × 16
	ETCA4	See note	8	16 levels	8 × 8
HILO format	HILO	R8 G8	16	-	8 × 8

Note: The ETC compressed format has compressed RGB components. ETCA4 has four alpha bits in addition to the RGB components.

The ETC format entails the lowest processing cost and is the fastest. The cost of processing other formats increases with the number of bits per texel.

The A4 or L4 format cannot be used together with any other format texture in a multi-texture.

5.2.1.1 RGB

These formats store RGB components. They do not store an alpha component. The following two types of formats exist.

Table 5-2 RGB Formats

Format	RGB Image	Alpha Image	Description
RGB565		-	This format uses 16 bits per texel. It uses 5 bits (32 levels) to represent the R component, 6 bits (64 levels) to represent the G component, and 5 bits (32 levels) to represent the B component.
RGB8		-	This format uses 24 bits per texel. All three RGB components can be represented using 8 bits (256 levels) each.

5.2.1.2 RGBA Formats

These formats store the RGB components and an alpha component. The following three types of formats exist.

Table 5-3 RGBA Formats

Format	RGB Image	Alpha Image	Description
RGBA4			This format uses 16 bits per texel. The RGBA components can all be represented using 4 bits (16 levels) each.
RGB5A1			This format uses 16 bits per texel. The three RGB components can be represented using 5 bits (32 levels) each and the alpha component can be represented using a single bit (transparent or not).
RGBA8			This format uses 32 bits per texel. The RGBA components can all be represented using 8 bits (256 levels) each. This has the greatest number of colors and the largest data size.

5.2.1.3 Alpha

These formats store only an alpha component. The following two types of formats exist. These are normally used as alpha values but they can also be used as grayscale colors.

Table 5-4 Alpha Formats

Format	RGB Image	Alpha Image	Description
A4	-		This format uses 4 bits per texel. The alpha component can be represented using 4 bits (16 levels).
A8	-		This format uses 8 bits per texel. The alpha component can be represented using 8 bits (256 levels).

5.2.1.4 Luminance

A grayscale format that stores only the brightness. The following two types of formats exist. These are normally used as grayscale values but they can also be used as alpha values.

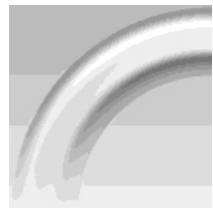
Table 5-5 Luminance Formats

Format	RGB Image	Alpha Image	Description
L4		-	This format uses 4 bits per texel. The brightness can be represented using 4 bits (16 levels).
L8		-	This format uses 8 bits per texel. The brightness can be represented using 8 bits (256 levels).

5.2.1.5 Luminance Alpha

These formats store the brightness and an alpha component. The following two types of formats exist.

Table 5-6 Luminance Alpha Formats

Format	RGB Image	Alpha Image	Description
LA4			This format uses 8 bits per texel. The brightness and alpha components can be represented using 4 bits (16 levels) each.
LA8			This format uses 16 bits per texel. The brightness and alpha components can be represented using 8 bits (256 levels) each.

5.2.1.6 ETC

This format stores RGB components that use compression. The ETC format uses a smaller data size than other texture formats.

Because this uses one-way compression, you may not be able to properly reproduce some images. Natural images and images with smooth color gradations are highly reproducible. In general, this is suitable for high-resolution images and images in which a single texel is not very significant.

Table 5-7 ETC Format

Format	RGB Image	Alpha Image	Description
ETC		-	With compression, this format uses 4 bits per texel. It compresses the RGB components to store them in four bits.

5.2.1.7 ETCA4

The ETCA4 format attaches alpha information to the ETC format. Its RGB components have the same characteristics as they do in the ETC format.

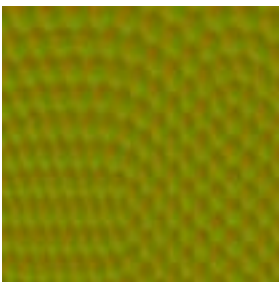
Table 5-8 ETCA4 Format

Format	RGB Image	Alpha Image	Description
ETCA4			With compression, this format uses 8 bits per texel. The RGB components are compressed and stored in 4 bits and the alpha component is represented using 4 bits (16 levels).

5.2.1.8 HILO

This is a texture format for normal mapping. You can apply normal mapping to polygons to simulate rough surfaces.

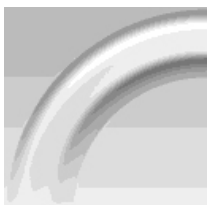
Table 5-9 HILO

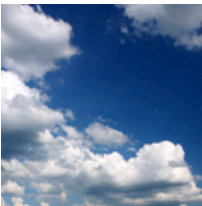
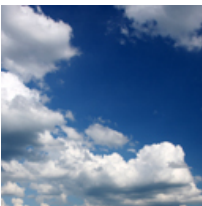
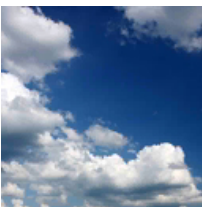
Format	Image	Description
HILO		This format is provided for normal mapping and uses 16 bits per texel. It maintains 8 bits of precision for the X and Y components of normal vector data in the R and G components, respectively.

5.2.2 Image List for Texture Formats

The following list shows the same (RGB) image in each texture format.

Table 5-10 RGB Image List for Texture Formats

Formats	RGB Images	
L4 LA4		

Formats	RGB Images	
L8 LA8		
RGBA4		
RGB5A1		
RGB565		
RGB8 RGBA8		
ETC ETCA4		

The following list shows the same (alpha) image in each texture format.

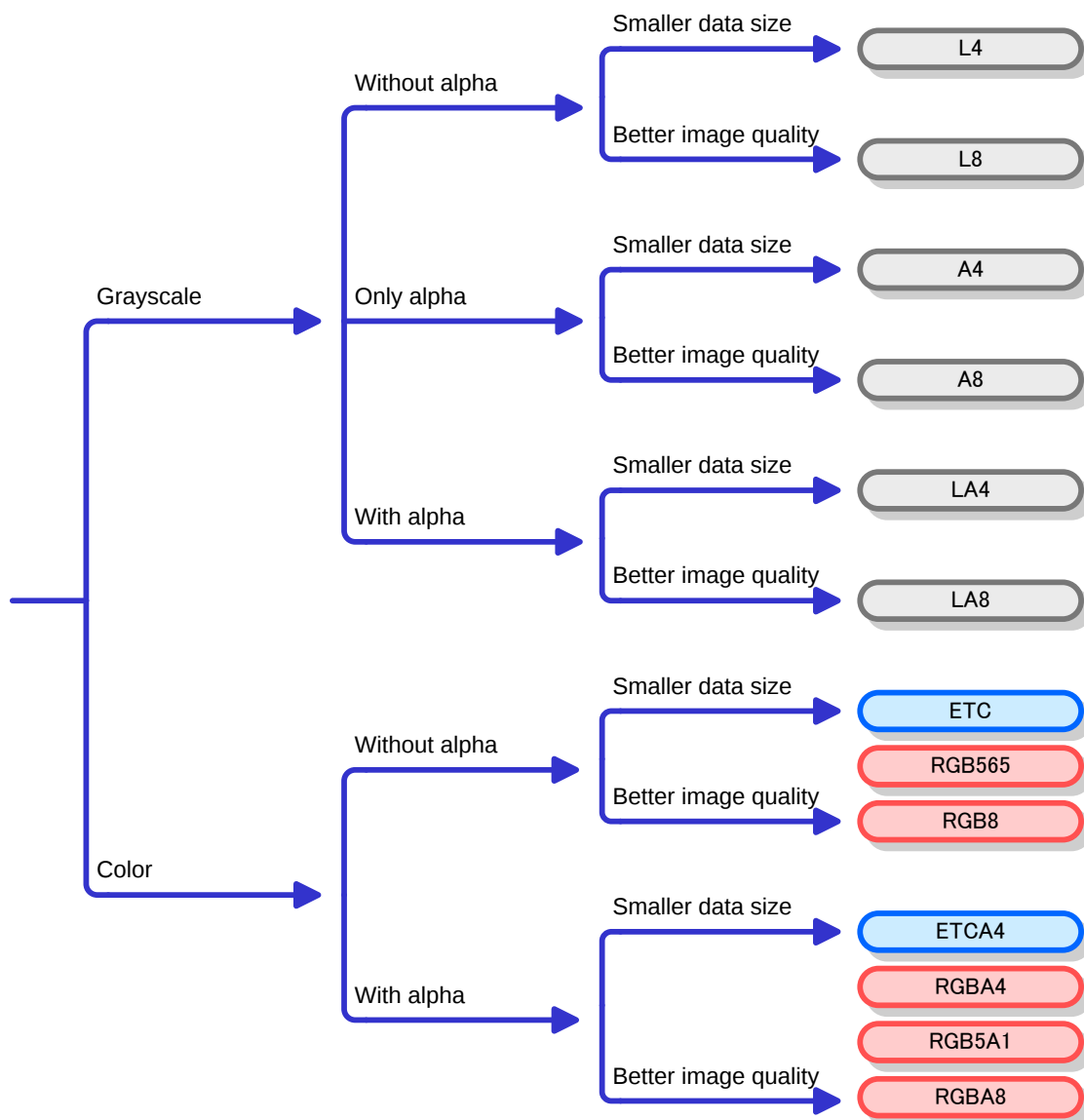
Table 5-11 Alpha Image List for Texture Formats

Formats	Alpha Images
RGB5A1	
RGBA4 A4 LA4 ETCA4	
RGBA8 A8 LA8	

5.2.3 How to Choose a Texture Format

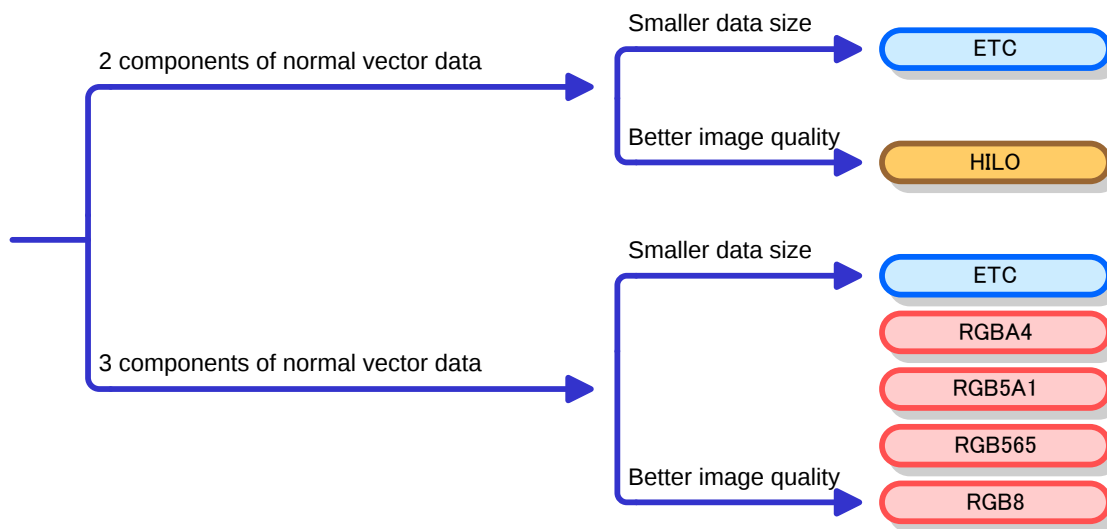
The following figure shows the process of choosing a texture format.

Figure 5-2 How to Choose a Texture Format



The following figure shows the process of choosing a texture format to apply as normal vector data for normal mapping. There are two types of normal mapping: one uses two texture components—R and G—as normal vector data and the other uses three texture components—R, G, and B—as normal vector data.

Figure 5-3 How to Choose a Texture Format for Normal Mapping

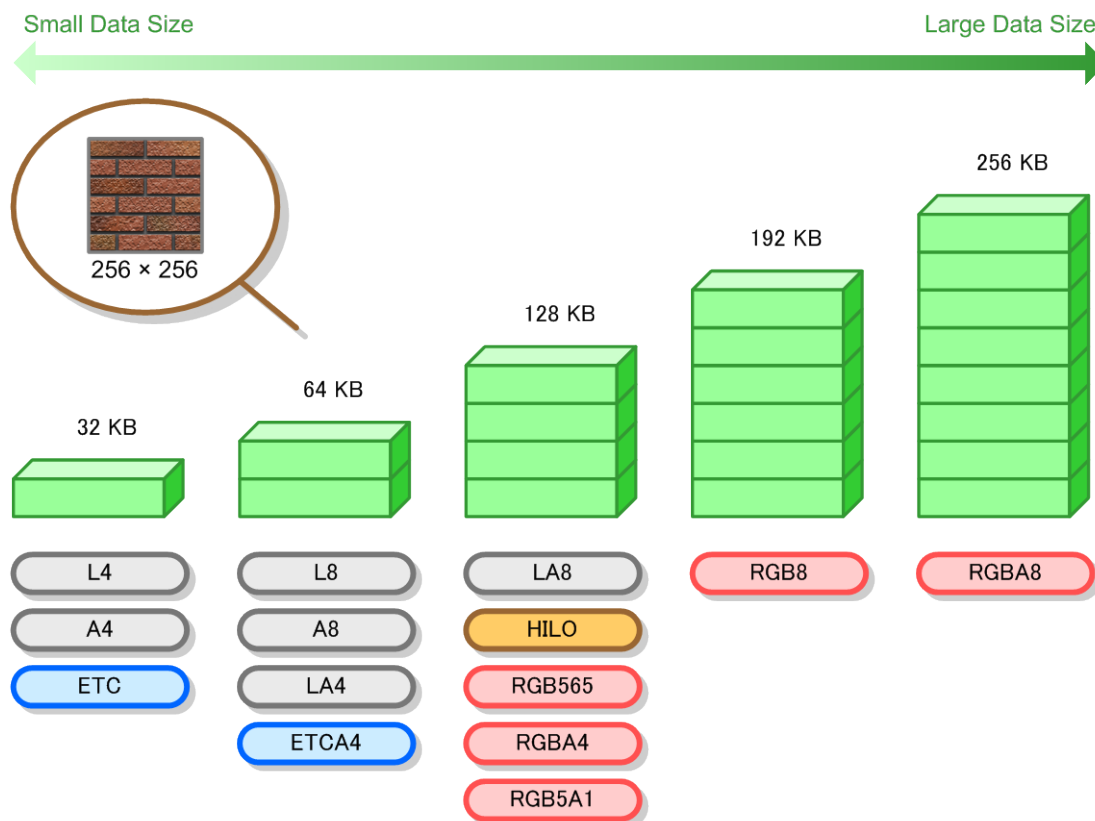


Note: If you use a texture format with only two components of normal vector data for normal mapping, the third normal component is calculated automatically. If you use a texture format other than HILO, the B component is ignored.

5.2.4 Texture Format Size Comparison

Texture data size is determined by the texture size (width by height) and the number of bits per texel.

For example, the following figure compares the memory usage of each format for a 256x256 texture. Memory usage is much more efficient when you choose a texture format in line with its intended use.

Figure 5-4 Texture Format Size Comparison (for a 256x256 Texture)

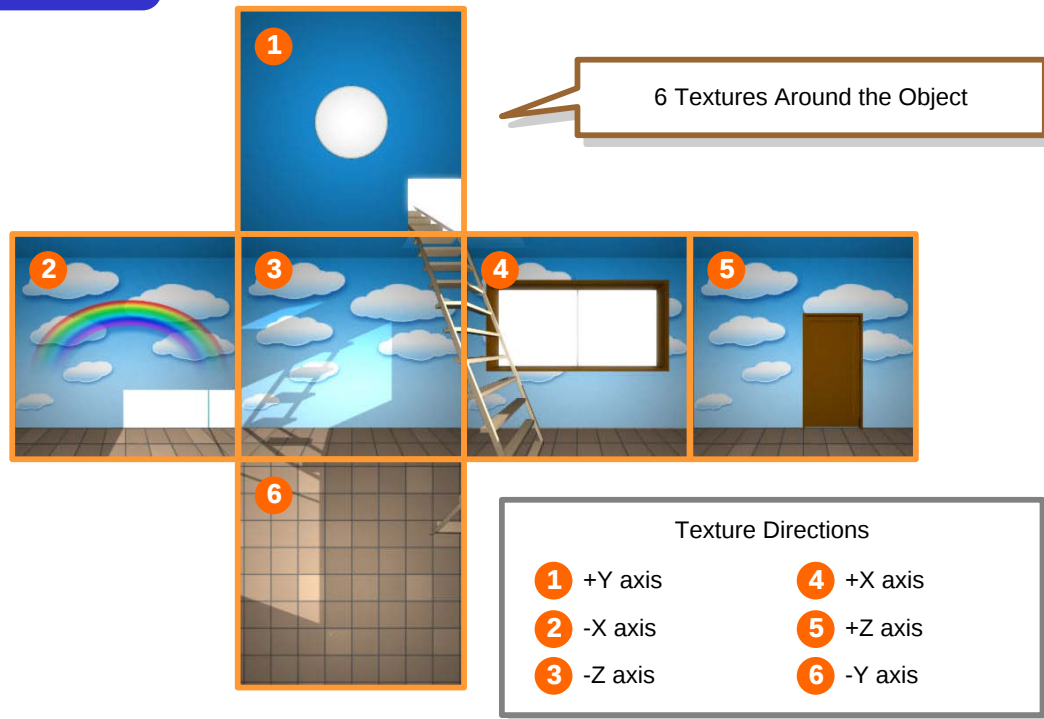
5.3 Cube Map Textures

A cube map texture is used for cube environment mapping. Cube environment mapping reflects a scene onto an object placed within it.

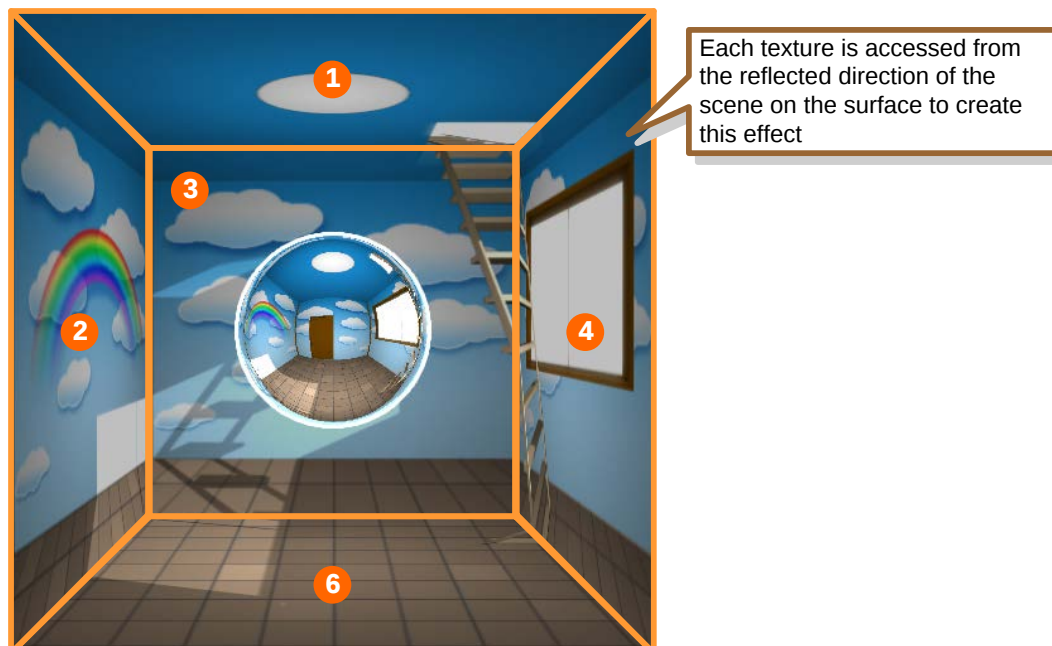
Cube map textures are required for each of the six directions around an object. A square texture must be used in each direction. Placed side-by-side, the textures look like an unfolded cube.

Figure 5-5 Cube Map Textures

Texture Structure



Applied Textures



5.4 Tiling Textures

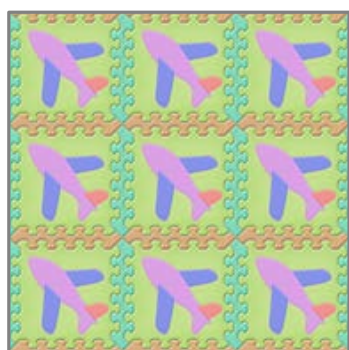
You can choose from the following four methods of tiling (repeating) textures to be mapped. Tiling is possible in both a texture's horizontal and vertical directions.

Table 5-12 Tiling Textures

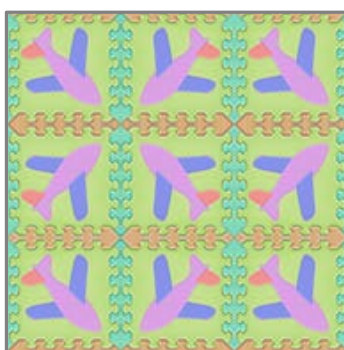
Tiling Method	Description
Repeat	Repeats the texture.
Mirror	Reverses and repeats the texture.
Clamping	Extends the colors at the edges of the texture.
Border Clamping	Fills the region around the texture with a specified border color.

The following figure shows examples of vertical and horizontal tiling for each of the tiling methods.

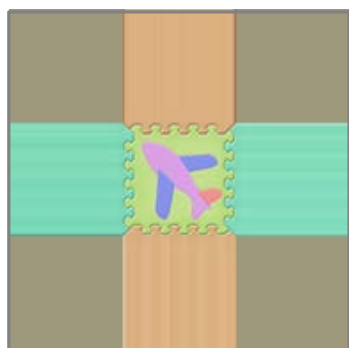
Figure 5-6 Tiling Textures



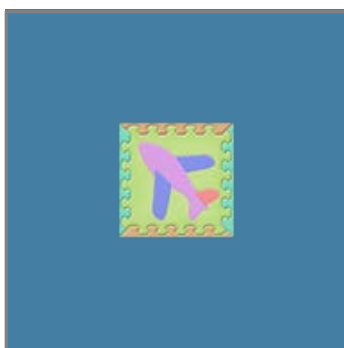
Repeat



Mirror



Clamping



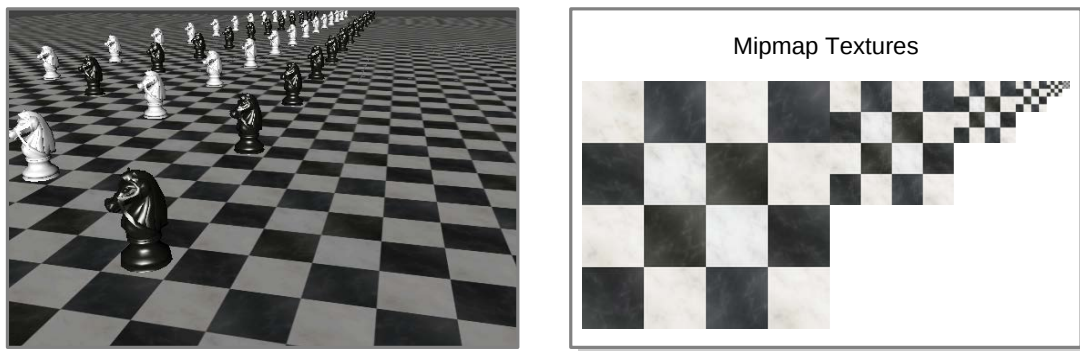
Border Clamping

Border Color

5.5 Mipmaps

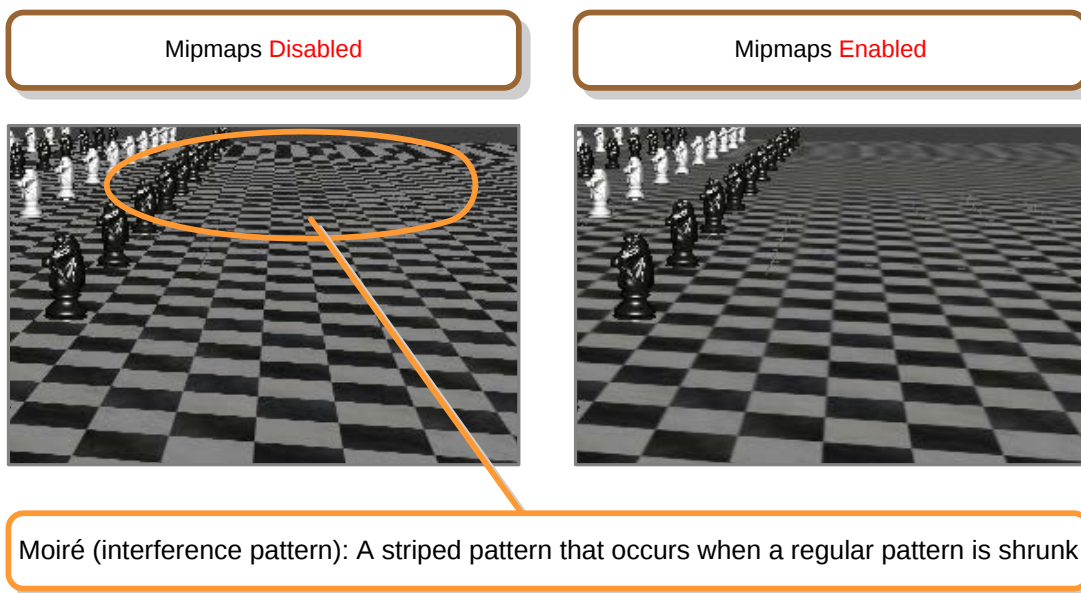
Mipmapping is a technique in which low-resolution textures are prepared in advance and then applied when displaying textures that have been shrunk.

Figure 5-7 Sample Application of Mipmaps



You can use mipmaps to reduce the processing load and avoid moiré patterns or other unnatural artifacts.

Figure 5-8 Example of Using Mipmaps to Remove Moiré Patterns

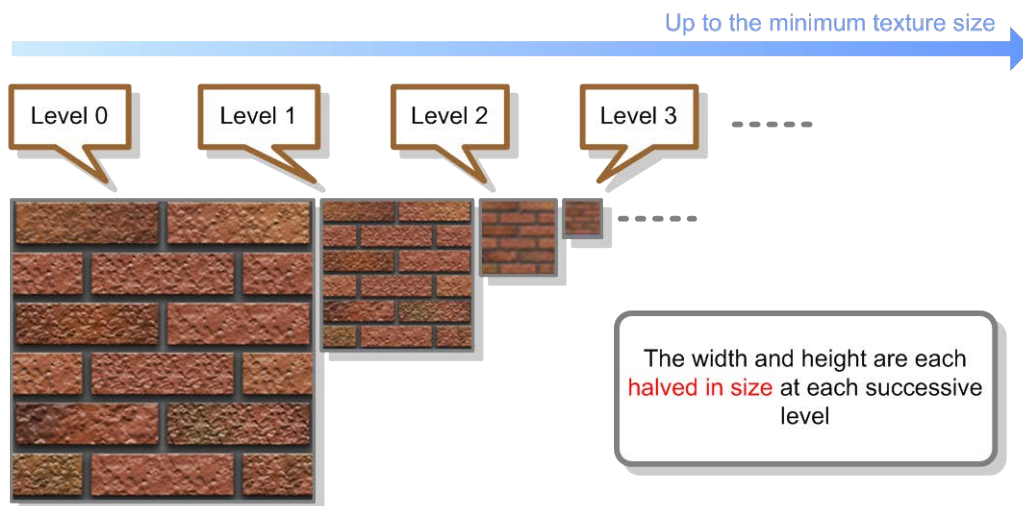


5.5.1 Mipmap Levels

Mipmap levels represent the texture stages used for mipmaps. Given level 0 as the size of the first texture, each successive level halves the width and height of the texture.

You can use up to eight mipmap levels until either the width or height reaches the smallest texture size.

Figure 5-9 Mipmap Levels



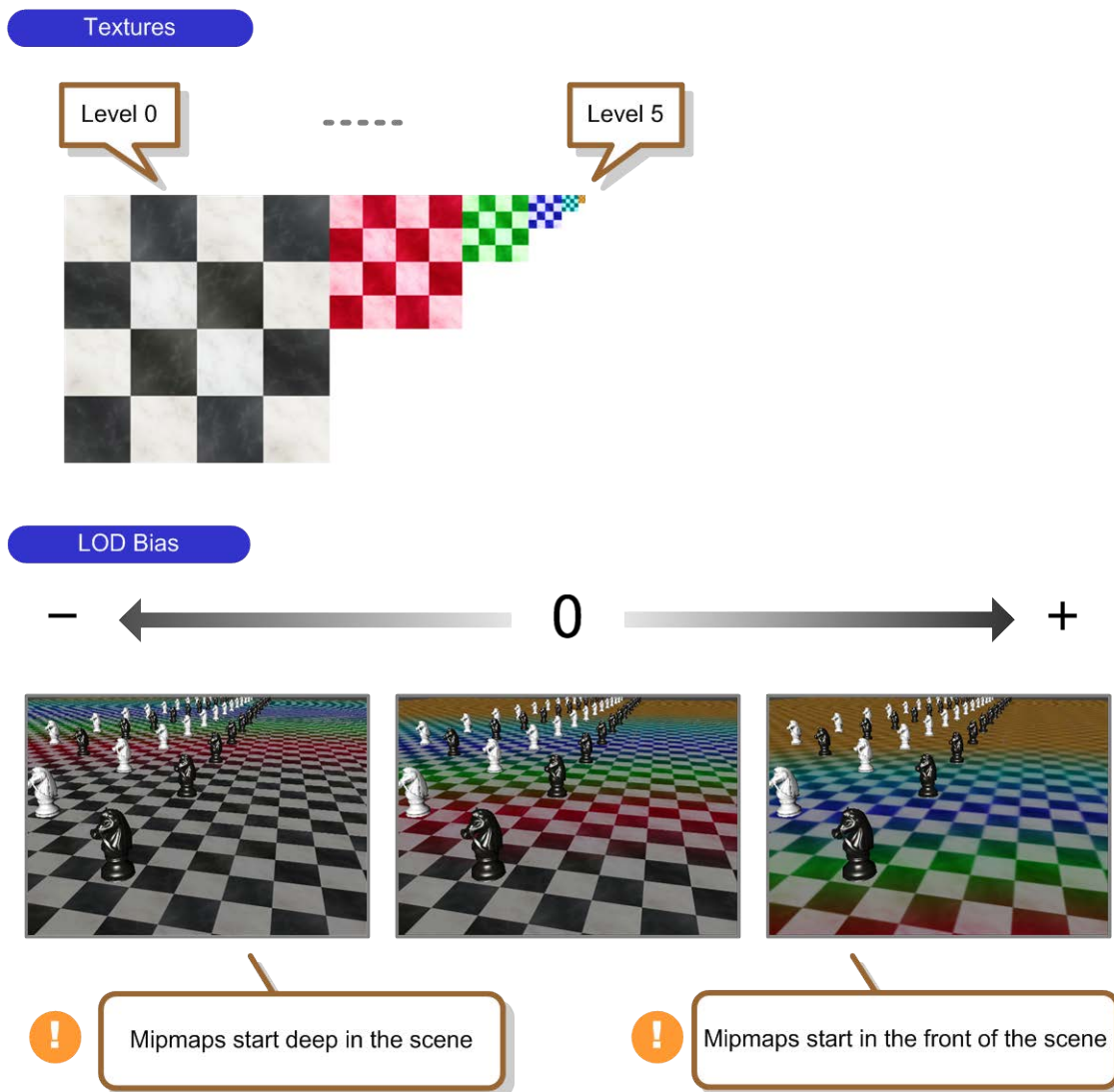
Note: The ETC format has a minimum texture size of 16x16, so it only has up to seven mipmap levels.

5.5.2 LOD Bias

Textures sometimes become overly blurred when mipmaps are used. This occurs when the applied texture LOD (level of detail) is larger than expected, causing textures from a low-resolution level to be referenced.

The LOD bias is added to the result of calculating the LOD and thus adjusts the mipmap level that is applied. The following figure uses a different texture color for each mipmap level to make the levels easier to see.

Figure 5-10 LOD Bias

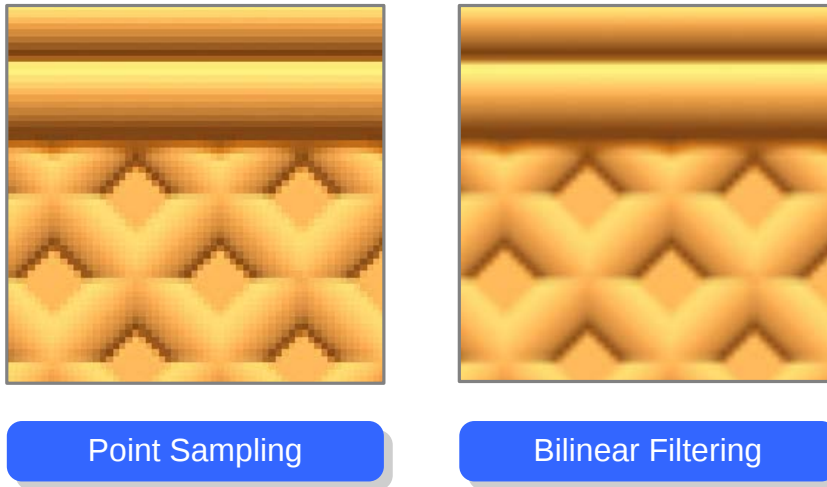


5.6 Texture Filters

Two types of filtering methods are used during texture mapping: point sampling and bilinear filtering.

Table 5-13 Texture Filters

Filtering Method	Description
Point Sampling	Determines a pixel's color by accessing its corresponding texel precisely. This displays distinct texels.
Bilinear filtering	Determines a pixel's color by interpolating the adjacent texel colors. This displays smooth textures.

Figure 5-11 Point Sampling and Bilinear Filtering

You can specify filters to use during both texture magnification and texture minification. If mipmaps are in use during texture minification, a texture filter can even be specified between mipmap levels.

Table 5-14 Texture Filtering (with Mipmaps in Use)

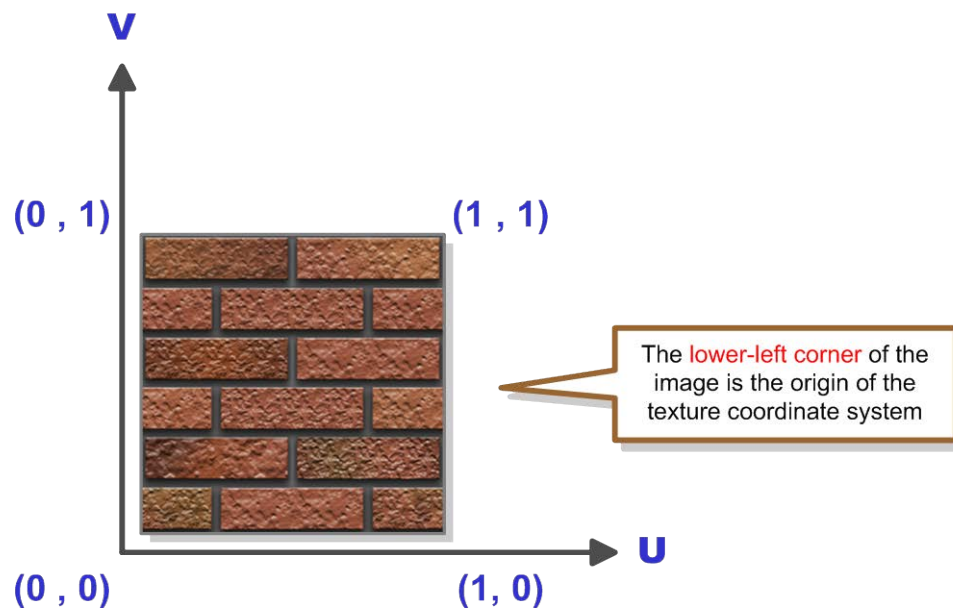
		Filtering Method Within the Same Level	
		Point Sampling	Bilinear Filtering
Filtering Method Between Mipmap Levels	Switch Quickly Between Levels		
Filtering Method Between Mipmap Levels	Interpolate Smoothly Between Levels		

Note: There is no difference in the cost to process filtering methods within the same level during texture magnification, but the filtering method that smoothly interpolates within the same level entails a higher cost during texture minification. There is a higher processing cost associated with a filtering method that smoothly interpolates between mipmap levels.

5.7 Texture Mapping

As shown in the following figure, the lower-left corner of an image is the origin for texture coordinates. The positive U axis extends to the right and the positive V axis extends up. Regardless of its size, a texture is mapped so that it fits precisely between 0.0 and 1.0 in both the U and V directions.

Figure 5-12 Texture Coordinate System



There are a number of ways to map textures, depending on how the textures are used.

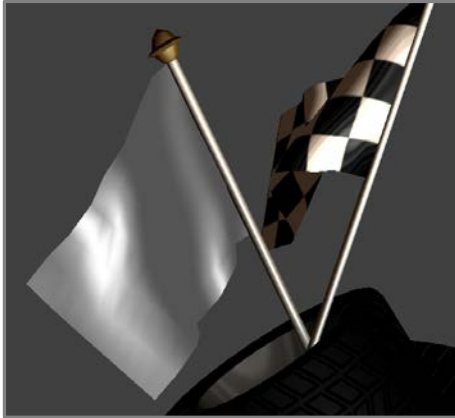
- UV mapping
- Projective mapping
- Cube environment mapping
- Normal mapping

5.7.1 UV Mapping

This method maps values based on the texture coordinates input as vertex attributes. It is the most common way to map textures.

Figure 5-13 UV Mapping

Object



Texture



Sample Application

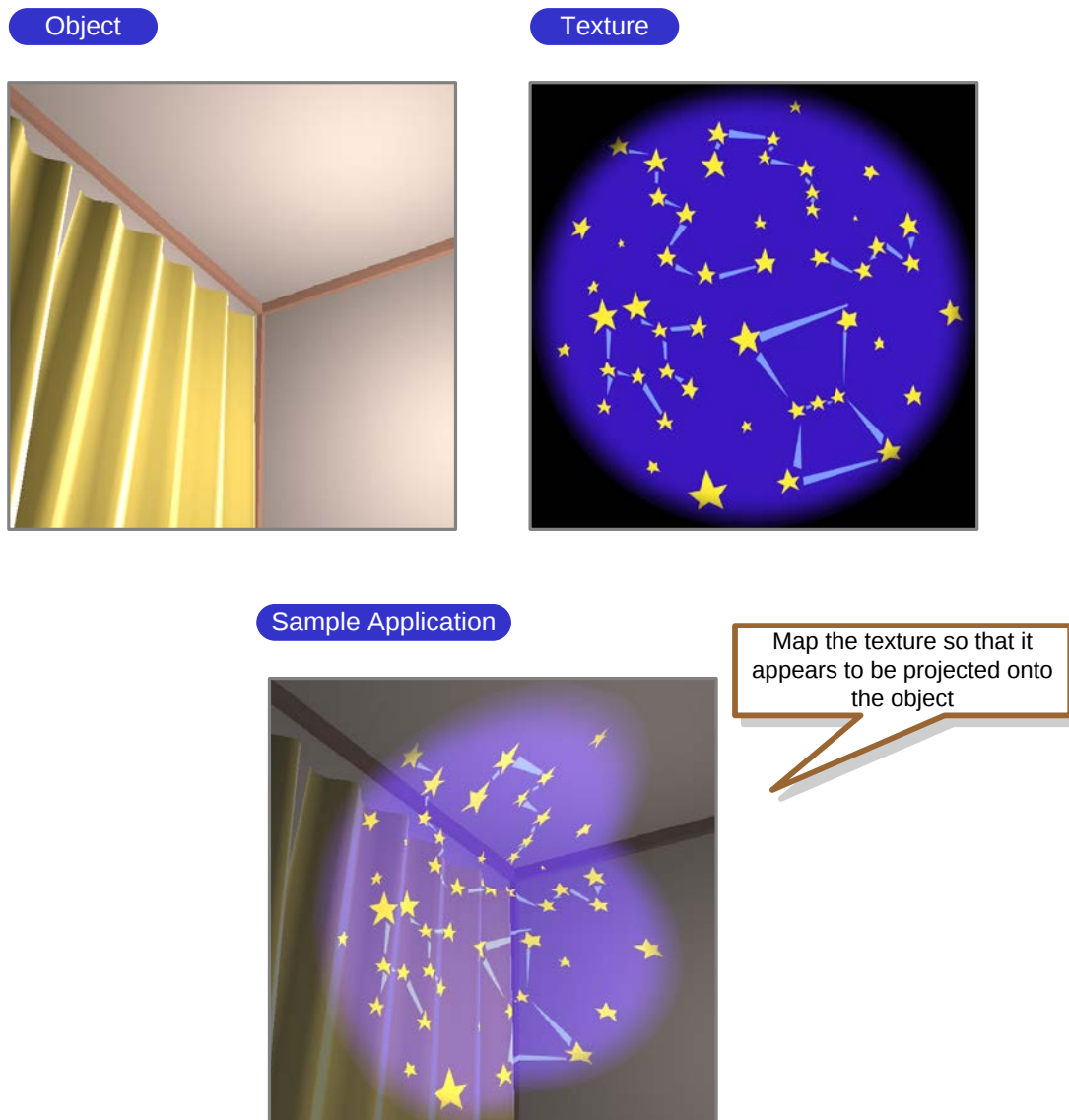


Map the texture based on its
(U,V) coordinates

5.7.2 Projective Mapping

This mapping method generates texture coordinates from an object's vertex coordinates. This allows you to depict spotlights, shadows, images projected as if by a slide projector, and so on.

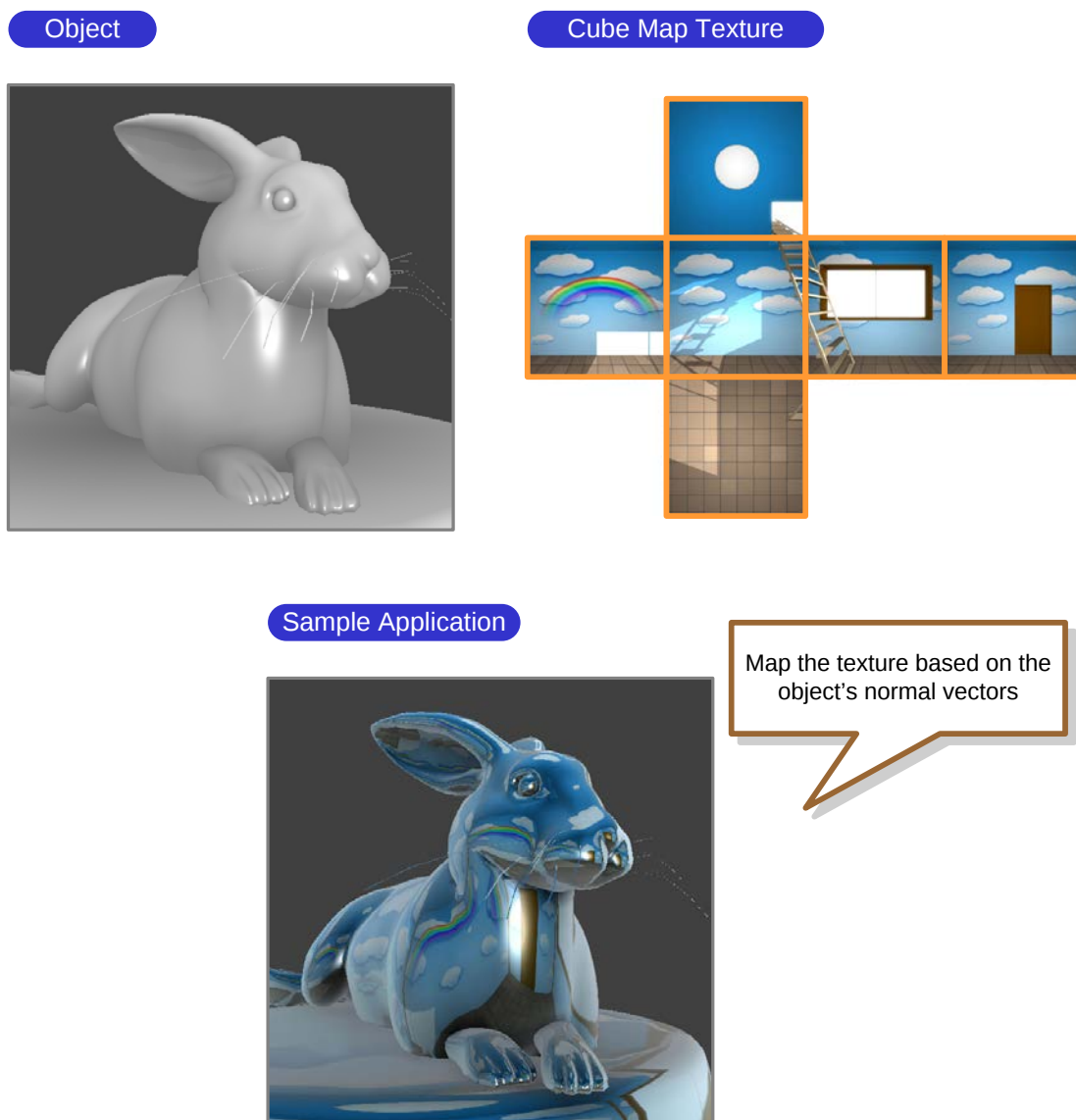
Figure 5-14 Projective Mapping



5.7.3 Cube Environment Mapping

This mapping method generates texture coordinates from an object's normal vectors and then uses those coordinates to look up values in a cube map texture. This can reflect a scene onto an object placed within it.

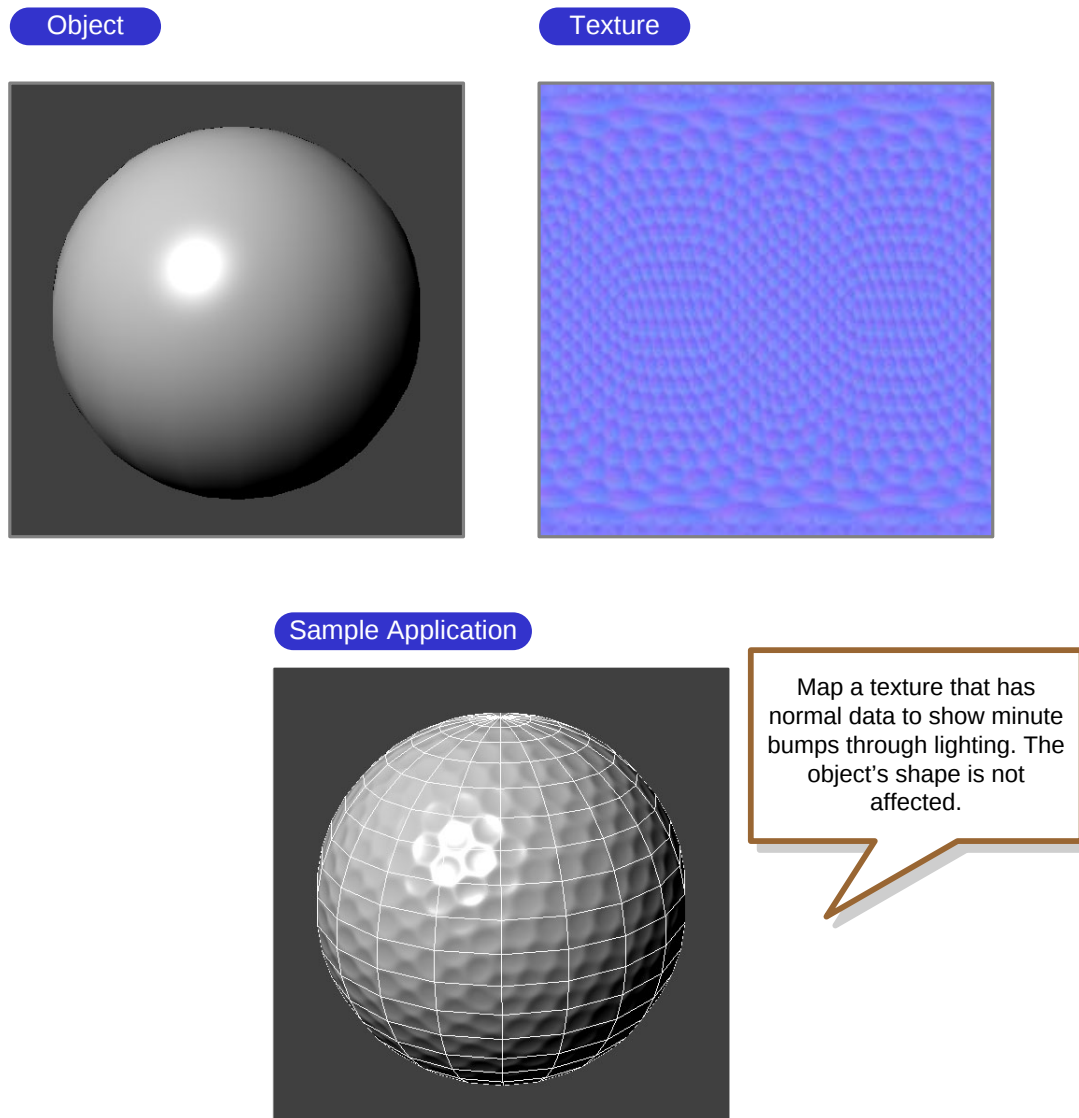
Figure 5-15 Cube Environment Mapping



5.7.4 Normal Mapping

This method maps a texture that has normal vector data and then runs fragment lighting based on the mapped normal vectors. This can simulate small bumps on an object.

Figure 5-16 Normal Mapping

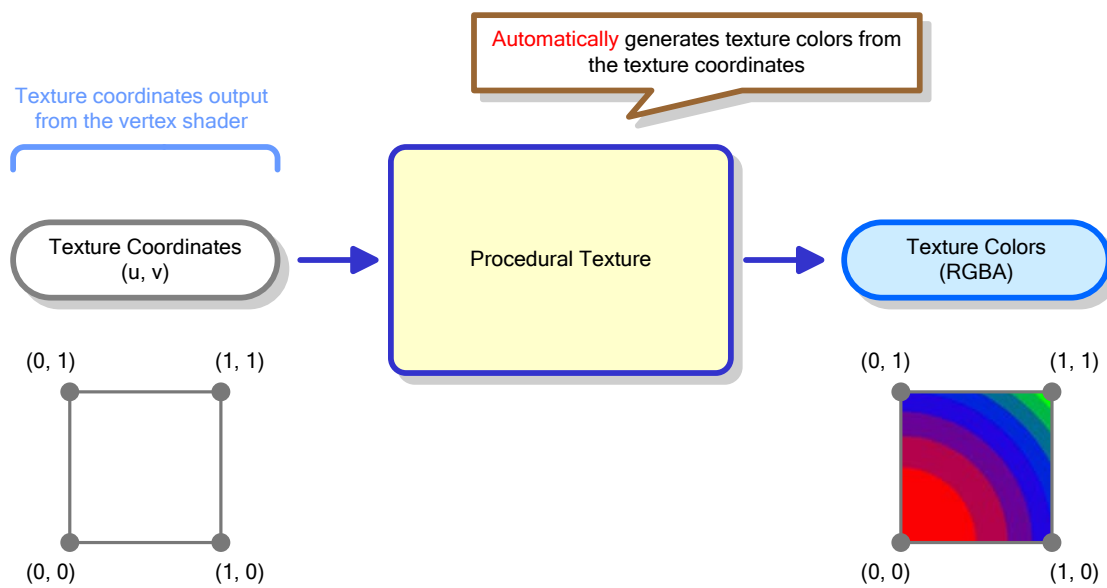


Note: Applying normal mapping after flipping a texture causes lighting to be applied inappropriately. For example, to render a convex and concave object that is symmetrical to left and right, you cannot just take half of the normals and flip them to use for the other half.

5.8 Procedural Textures

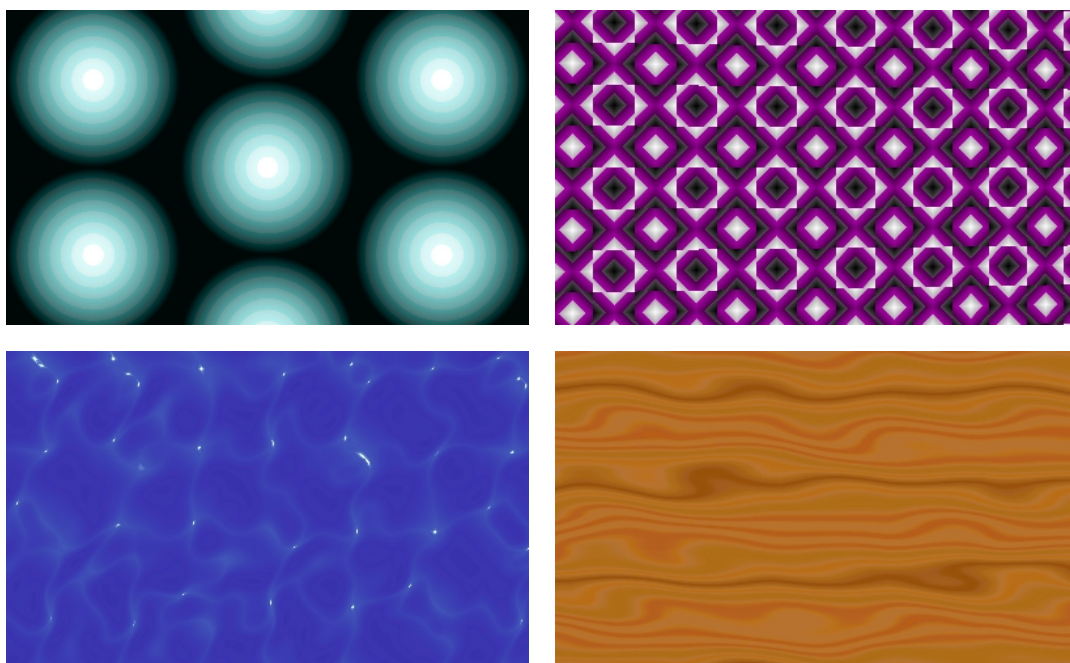
Unlike normal textures that require texture images, procedural textures automatically generate the texture image from the configured parameters. Texture colors are calculated based on the texture coordinates (u, v) output from the vertex shader.

Figure 5-17 Procedural Textures



Procedural textures are good for expressing geometrical and random patterns.

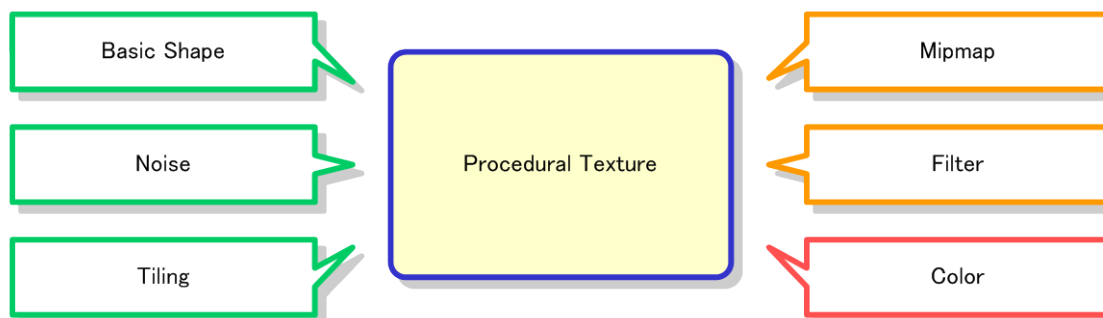
Figure 5-18 Procedural Texture Examples



5.8.1 Configuring Procedural Textures

Broadly speaking, procedural textures have the following six components, and the final texture is generated after each of these is properly configured.

Figure 5-19 Configuring Procedural Textures



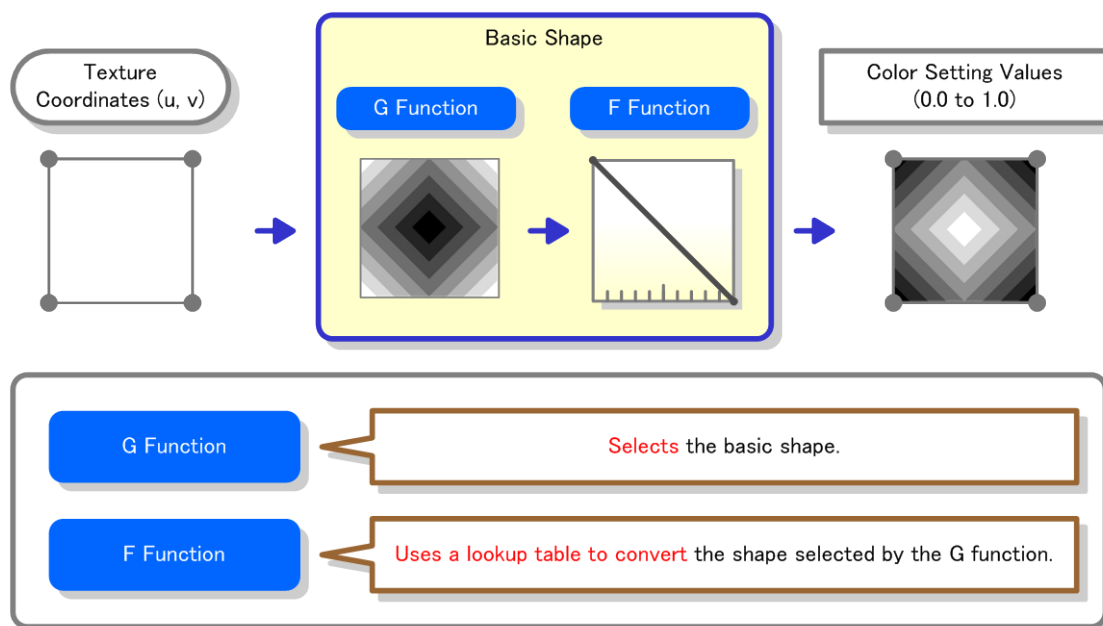
5.8.2 Basic Shape

This parameter configures the basic shape of the procedural texture.

The basic shape is processed in two stages, configured using the G and F functions. Use the G function to select the basic shape, and the F function to convert that value.

Basic shape processing converts texture coordinates (u, v) to values used to configure the color (0.0 through 1.0).

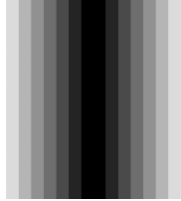
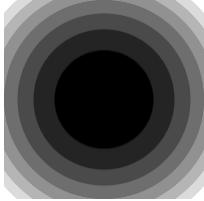
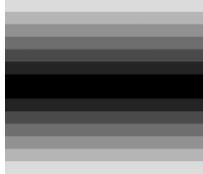
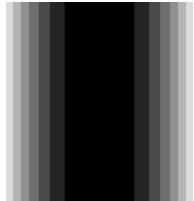
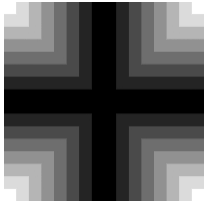
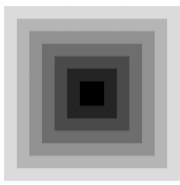
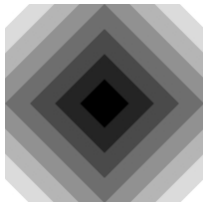
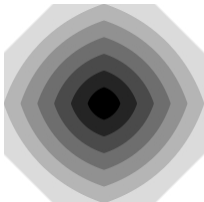
Figure 5-20 Basic Shape

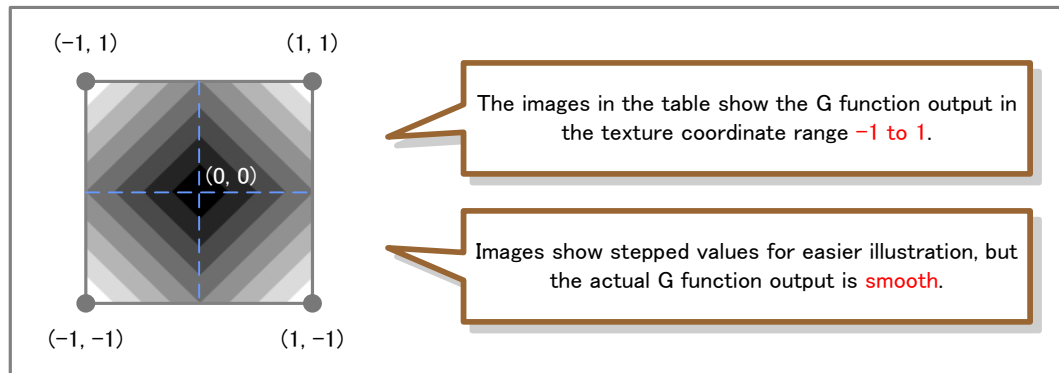


Note: You can select separate shapes (G and F functions) for the color (RGB) and alpha (A).

You can select from the following 10 types for the G function.

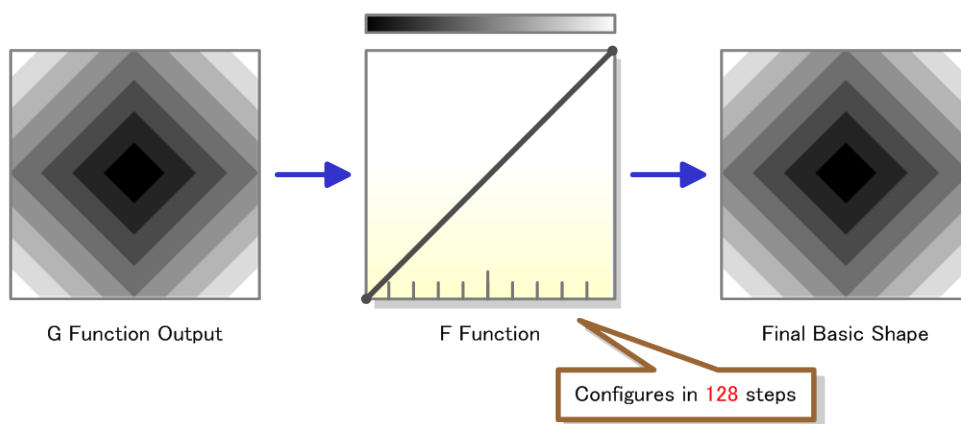
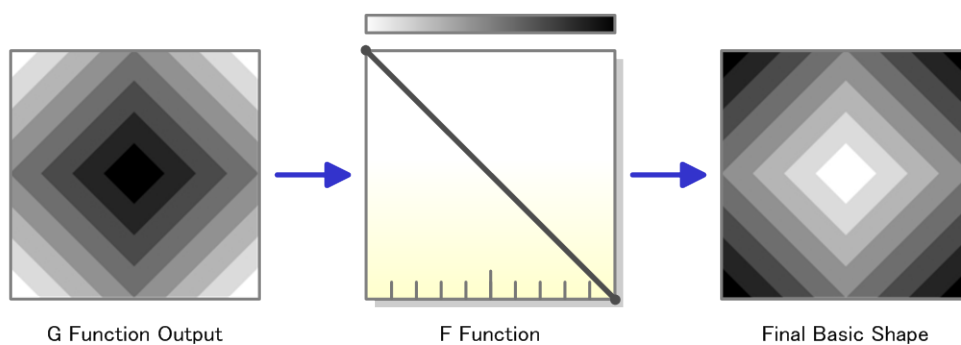
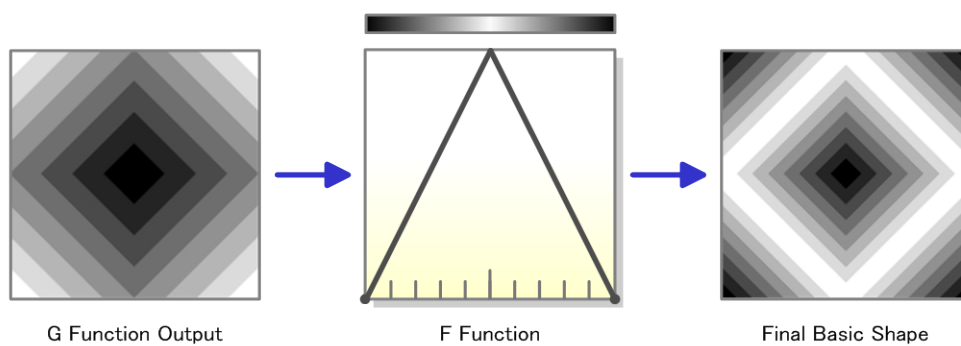
Table 5-15 Basic Shape (G Function)

Basic Shape	Image	Basic Shape	Image
U		ADD2	
V		ADDSQRT2	
U2		MIN	
V2		MAX	
ADD		RMAX	



The values in the shape selected by the G function can be converted using the F function. The F function configures values using a 128-step lookup table.

Figure 5-21 Basic Shape (F Function)

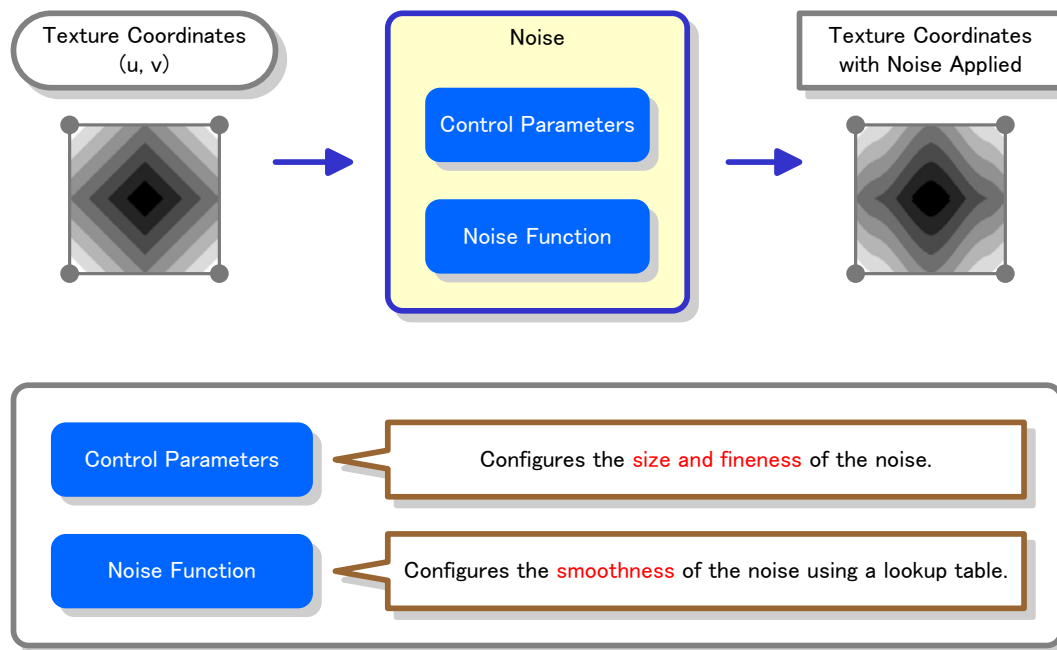
Example 1 No conversion**Example 2** Invert shape**Example 3** Convert to finer gradations

5.8.3 Noise

This configures noise to apply to the basic shape. Apply noise to generate images with a more varied and natural appearance.

You can configure control parameters that determine the size and fineness of the noise, and a noise function that controls the smoothness of the noise.

Figure 5-22 Noise

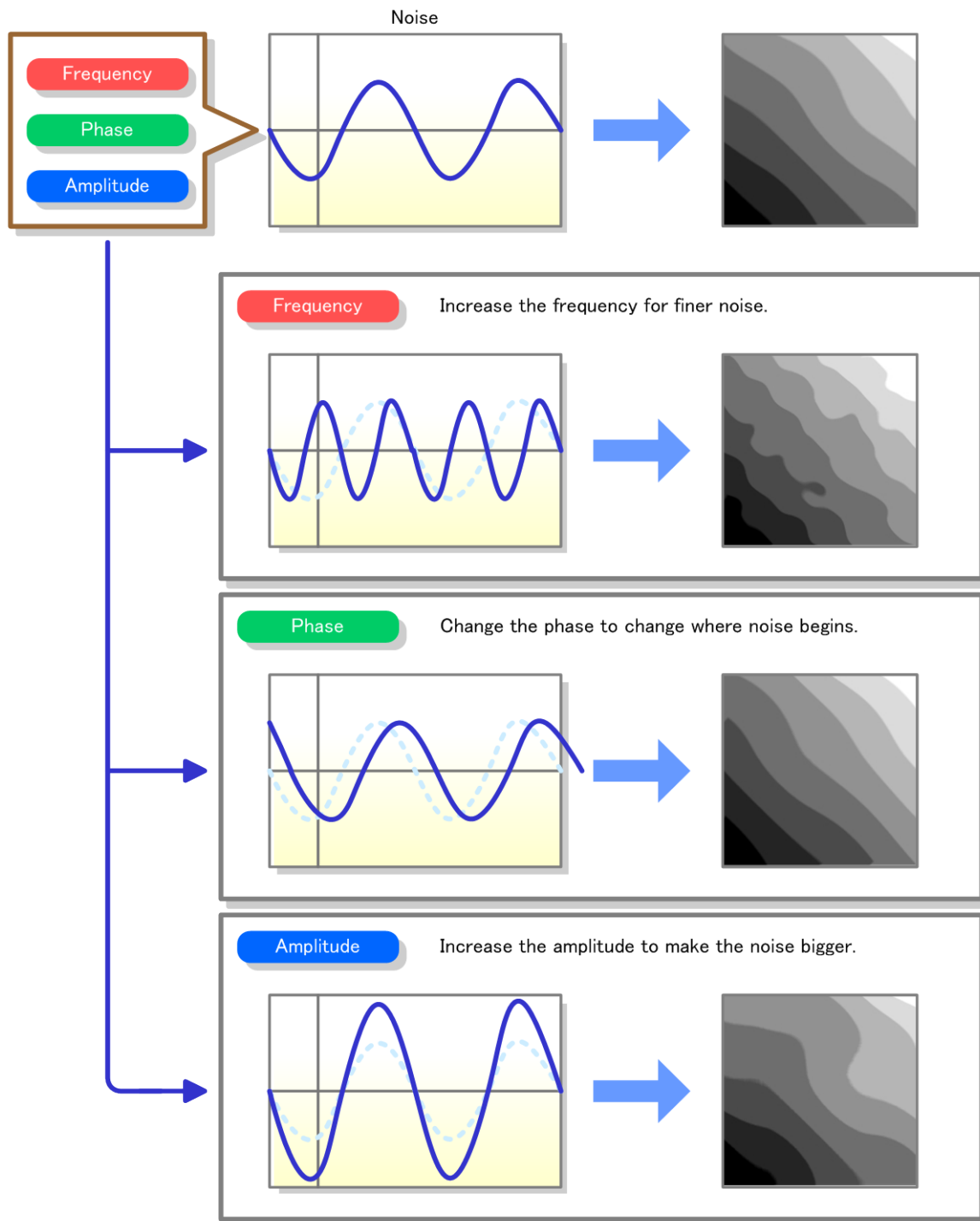


You can configure the following three noise control parameters independently for the vertical and horizontal directions.

Table 5-16 Control Parameters

Parameter	Description
Frequency	Noise fineness
Phase	Noise start position
Amplitude	Noise size

Figure 5-23 Noise Control Parameters

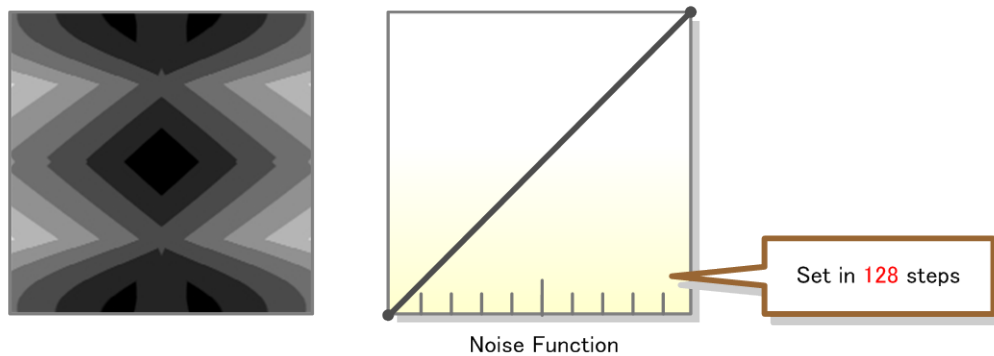


Note: Gradually change the phase to create an animation of the noise flowing.

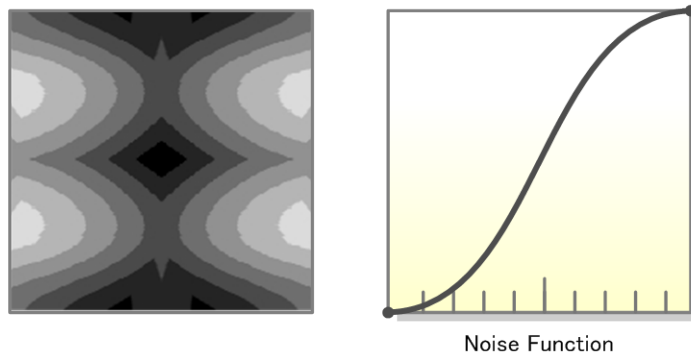
Noise functions are configured with a 128-step lookup table. Set the table with values evenly progressing from 0 to 1 for overall smooth noise. Set the table with discontinuous values for discontinuous noise, giving the appearance of boundaries.

Figure 5-24 Noise Functions

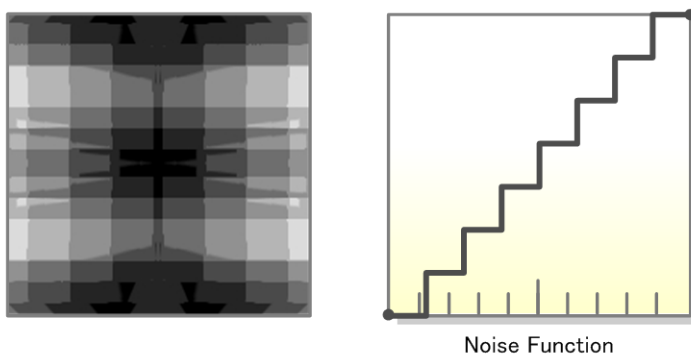
Example 1 Linear noise function going from 0 to 1



Example 2 Curved noise function going from 0 to 1



Example 3 Stepped noise function going from 0 to 1

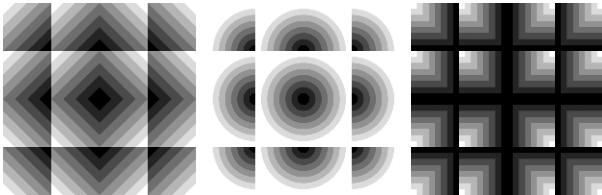
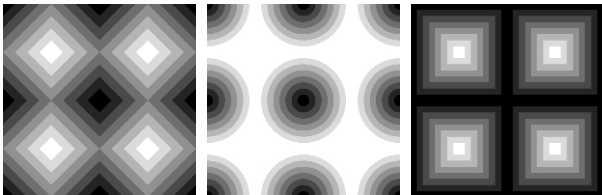
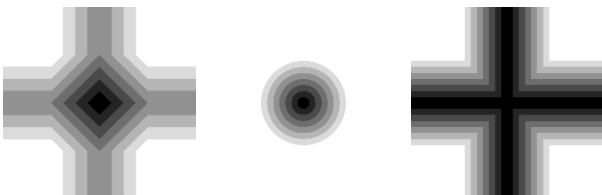
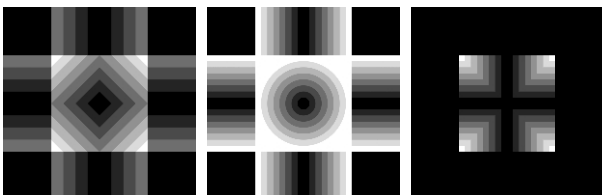
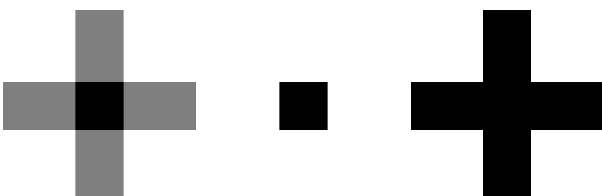


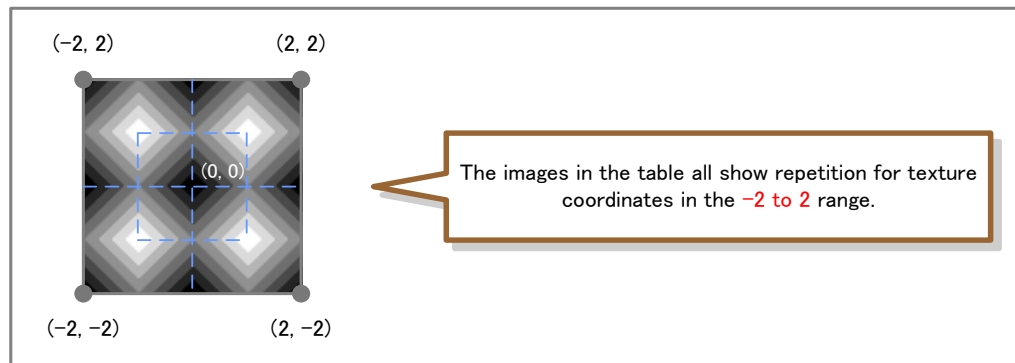
Note: The same lookup table is used by a noise function for both the vertical and horizontal directions.

5.8.4 Tiling

This parameter configures how a texture is repeated. You can configure the repetition method and the shift to use when repeating a texture independently for both the vertical and horizontal directions. Choose from the following five repetition methods.

Table 5-17 Repetition Methods

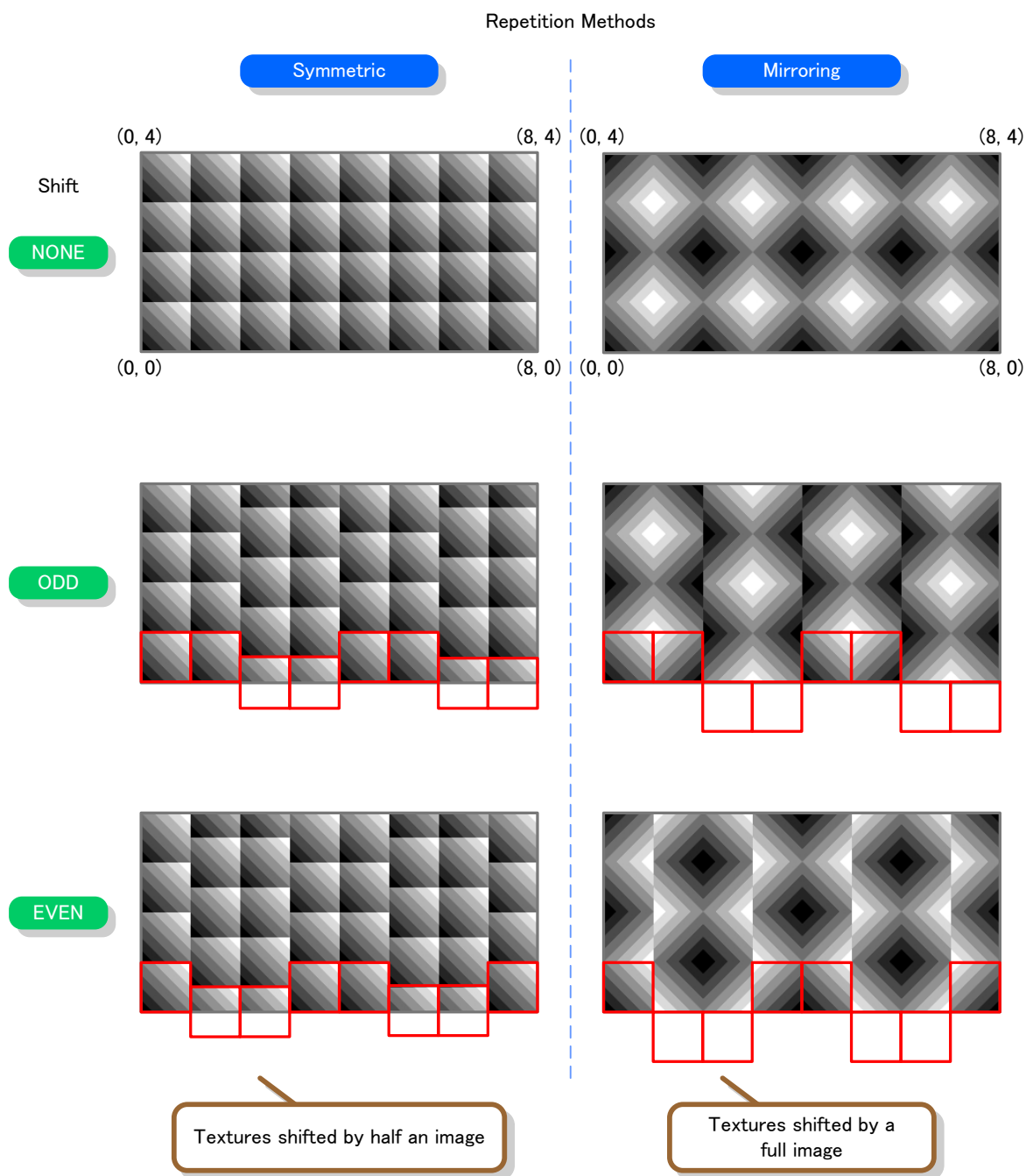
Repetition Method	Description	Image
Symmetric	Repeats the texture as is.	
Mirroring	Repeats the texture while flipping it.	
Edge Clamping	Clamps the texture coordinates at 1.	
Zero Clamping	Clamps the texture coordinates at 0.	
Pulsing	Converts texture coordinate values under 0.5 to 0, and values of 0.5 or higher to 1.	



Depending on the shift setting, you can repeat a texture image while shifting by specific texture coordinates. You can select no shift, a shift by odd values, or a shift by even values, specifying independently for both the vertical and horizontal directions.

The degree of shift differs depending on the repetition method. With mirroring, texture images are shifted by a full image, but for other repetition methods, texture images are shifted by only half an image. The figure below shows shift just in the vertical direction.

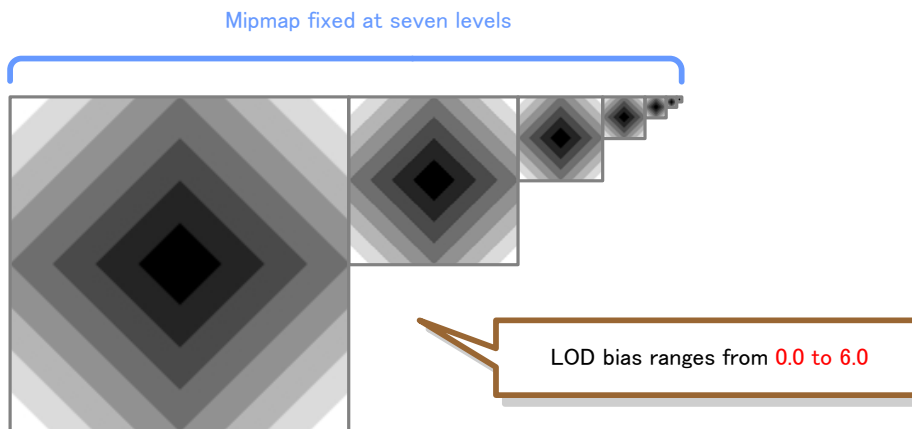
Figure 5-25 Shift



5.8.5 Mipmap

Procedural textures support seven-level fixed mipmaps, and you can set the LOD bias from 0.0 to 6.0.

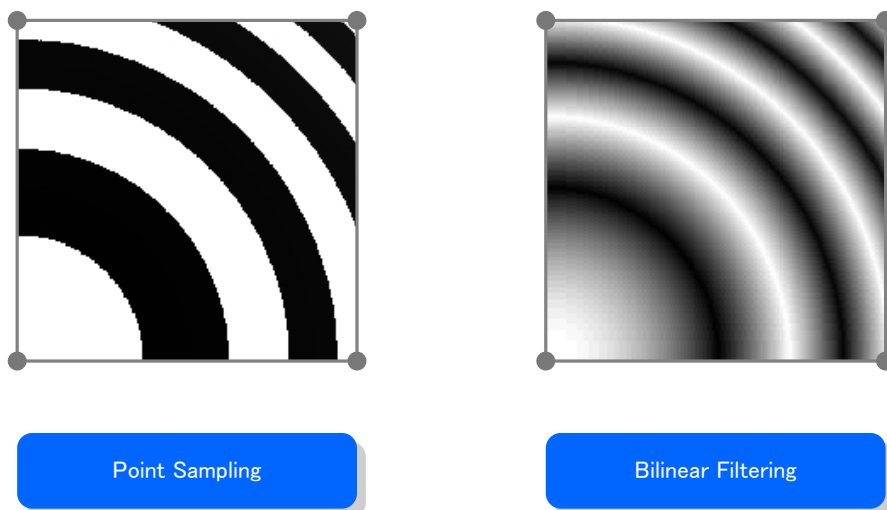
Figure 5-26 Shift



5.8.6 Filter

For procedural textures, you can configure the filtering method when shrinking to either point sampling or bilinear filtering. When mipmapping, you can select the filtering method to use between mipmap levels.

Figure 5-27 Filter

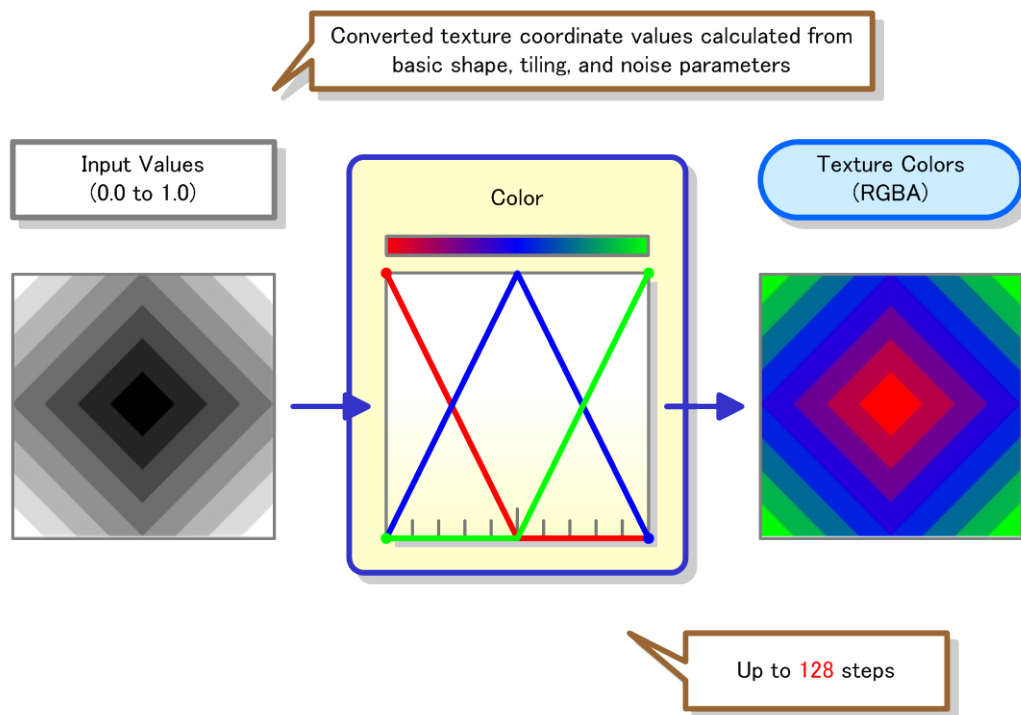


5.8.7 Color

This parameter configures the final colors of the procedural texture. Set the values for each of the RGBA components in a lookup table. You can set the number of steps in the lookup table (the number of colors configured) as 128, 64, 32, 16, 8, 4, 2, or 1.

When using mipmaps, you must configure lookup tables for each of the seven levels. Each of these lookup tables can have 128, 64, 32, 16, 8, 4, or 2 steps.

Figure 5-28 Color



Note: If the basic shape has color and alpha configured separately, the alpha lookup table is ignored.

5.8.8 Comparing with Normal Textures

The following table shows the differences in features provided by procedural and normal textures.

Table 5-18 Features of Procedural and Normal Textures

Feature	Procedural Texture	Normal Texture
Maximum texture size	Approx. 4000 texels	1024 texels
Minimum texture size	1 texel	8 texels
Expansion filtering	No	Yes
Shrink filtering	Yes	Yes
Mipmaps	Yes	Yes
Mipmap levels	Fixed at 7	Up to 8
Filtering between mipmap levels	Yes	Yes
LOD bias	0.0 to 6.0	Any

Procedural textures have the following performance characteristics.

- Lower processing load than normal textures.
- Same processing load regardless of procedural texture settings.
- All require the same amount of memory.

5.9 Usable Texture Combinations

You can register up to four textures, textures 0–3, simultaneously on a CTR system. The combinations of mapping methods that can be applied to each texture are restricted as follows.

Table 5-19 Usable Texture Combinations

	Texture 0	Texture 1	Texture 2	Texture 3	Maximum Number of Textures
UV mapping	✓	✓	✓	✓	4
Projective mapping	✓	✓	✓	✓	4
Cube environment mapping	✓	-	-	-	1
Normal Mapping	✓	✓	✓	✓	1

The following restrictions apply to the combinations of textures in use and texture coordinates output from the vertex shader.

Table 5-20 Combinations of Textures and Texture Coordinates

	Texture 0	Texture 1	Texture 2	Texture 3
Texture coordinate 0	✓	-	-	✓
Texture coordinate 1	-	✓	✓	✓
Texture coordinate 2	-	-	✓	✓

Note: Texture 3 is used exclusively for procedural textures.

Because the vertex shader can only output up to three types of texture coordinates, if you use four textures at once some of them must share the same texture coordinates.

Processing cost increases with the number of textures registered at one time.

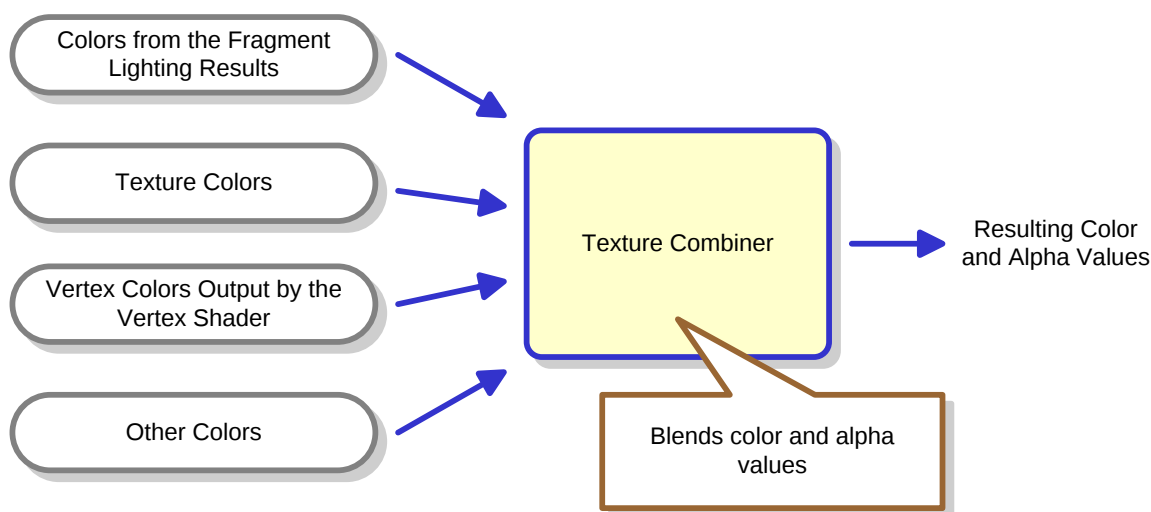
6 Texture Combiners

6.1 What Are Texture Combiners?

Texture combiners are a mechanism for blending input, such as the results of fragment lighting and textures, to determine the final color and alpha values.

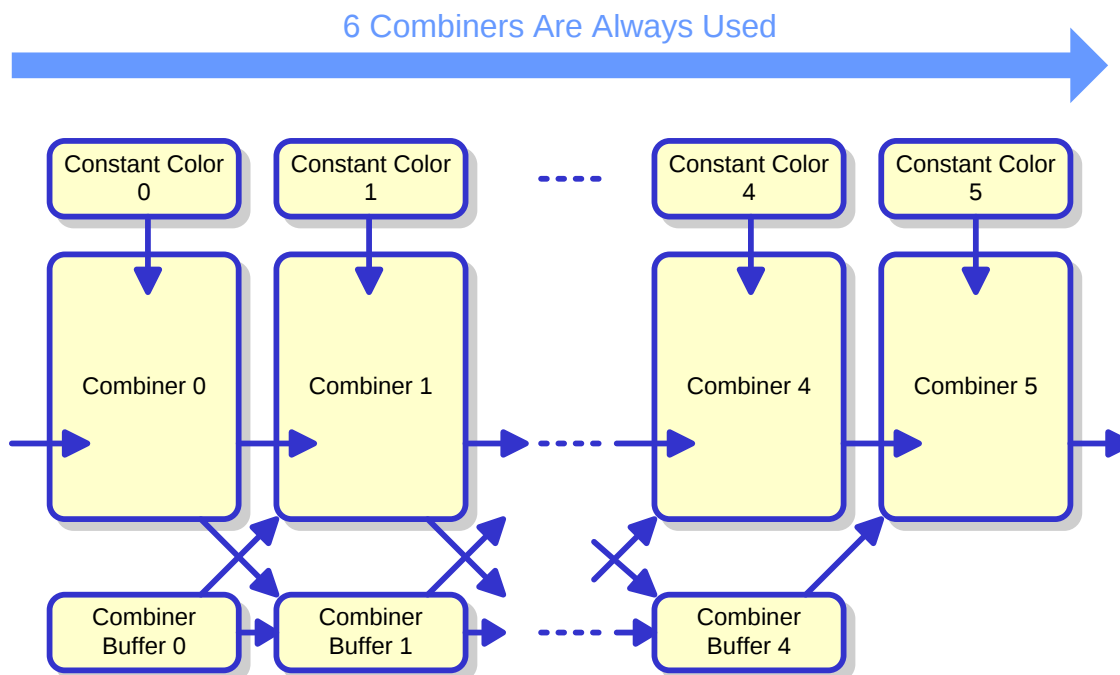
By blending lit colors and texture colors, you can combine multiple textures and depict a variety of materials.

Figure 6-1 What Are Texture Combiners?



6.2 Texture Combiner Structure

There are six texture combiner stages for blending colors. A constant color can be given as input and a combiner buffer is connected to each texture combiner stage (details on constant colors and combiner buffers are given in sections 6.3.5 Constant Colors and 6.3.6 Combiner Buffers).

Figure 6-2 Texture Combiner Structure

Note: Combiner buffer 5 does not exist.

6.3 Texture Combiner Settings

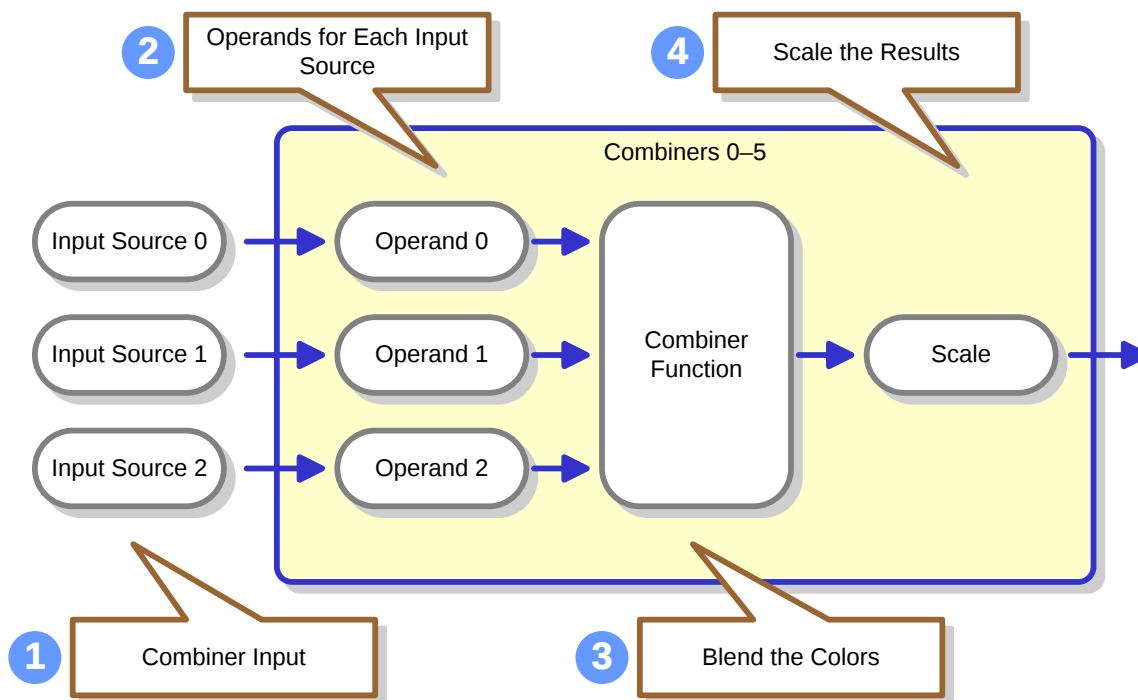
Each texture combiner stage processes colors and alpha values independently. The colors and alpha values each have the following four settings.

Table 6-1 Texture Combiner Settings

Setting	Description
Input source	Selects three inputs to a texture combiner.
Operands	Selects which components of the input source to use.
Combiner function	Selects the method for blending the input sources.
Scaling	Selects the scale to be applied to the blended results.

The following operations are run at each texture combiner stage. The output of each texture combiner stage is clamped between 0.0 and 1.0.

Figure 6-3 Texture Combiner Settings



Note: A texture combiner's processing load is always fixed, regardless of its settings. For example, a texture combiner's blending operations have the same processing load no matter how many times a texture is selected as an input source in each stage.

6.3.1 Input Sources

Three color and three alpha input sources are specified to each texture combiner stage. You can choose from the following 10 input sources.

Table 6-2 Input Sources

Input Sources	Description
Previous output color	Color output from the previous texture combiner stage. This cannot be used by texture combiner 0.
Vertex color	Color output from the vertex shader. The color is linearly interpolated across each polygon vertex and then applied.
Primary color	A result (output) of fragment lighting.
Secondary color	A result (output) of fragment lighting.
Texture color 0	The color of the texture registered as texture 0.
Texture color 1	The color of the texture registered as texture 1.
Texture color 2	The color of the texture registered as texture 2.
Texture color 3	The color of texture 3 (the procedural texture).

Input Sources	Description
Combiner buffer color	Color output from the previous combiner buffer stage. This cannot be used by texture combiner 0.
Constant color	A color that can be freely set at each texture combiner stage.

Note: Of the input sources starting with texture combiner 1, you must select at least one of the following input sources: the color output from the previous stage, the constant color, or the combiner buffer color.

6.3.2 Operands

Texture combiners allow you to specify which input source components to pass to combiner functions. These are called *operands*. You can select the following operands for each color and alpha input.

Color operands also allow you to swap (exchange) their color components.

Table 6-3 Color Operands

Color Operands	Color (R, G, B) to Which Operands Have Been Applied
RGB	(R, G, B)
A	(A, A, A)
R	(R, R, R)
G	(G, G, G)
B	(B, B, B)
1 - RGB	(1 - R, 1 - G, 1 - B)
1 - A	(1 - A, 1 - A, 1 - A)
1 - R	(1 - R, 1 - R, 1 - R)
1 - G	(1 - G, 1 - G, 1 - G)
1 - B	(1 - B, 1 - B, 1 - B)

Table 6-4 Alpha Operands

Alpha Operands	Alpha (A) to Which Operands Have Been Applied
A	A
R	R
G	G
B	B
1 - A	1 - A
1 - R	1 - R

Alpha Operands	Alpha (A) to Which Operands Have Been Applied
1 - G	1 - G
1 - B	1 - B

6.3.3 Combiner Functions

A *combiner function* is an equation that blends input values to which operands have been applied. You can specify both a color and an alpha combiner function. You can choose from the following 10 types of combiner functions.

The table uses **A**, **B**, and **C** to represent input sources 0, 1, and 2, respectively, after operands have been specified.

Table 6-5 Combiner Functions

Combiner Function	Equation	Description
REPLACE	A	Outputs A unchanged.
MODULATE	$\mathbf{A} \times \mathbf{B}$	Multiplies A and B .
ADD	$\mathbf{A} + \mathbf{B}$	Adds A and B .
ADD_SIGNED	$\mathbf{A} + \mathbf{B} - 0.5$	Adds A and B and then subtracts 0.5.
INTERPOLATE	$\mathbf{A} \times \mathbf{C} + \mathbf{B} \times (1 - \mathbf{C})$	Blends A and B using the ratio C .
SUBTRACT	$\mathbf{A} - \mathbf{B}$	Subtracts B from A .
ADD_MULT	$(\mathbf{A} + \mathbf{B}) \times \mathbf{C}$	Adds A and B and then multiplies them by C .
MULT_ADD	$(\mathbf{A} \times \mathbf{B}) + \mathbf{C}$	Multiplies A and B and then adds C .
DOT3_RGB	Calculates the dot product of the vectors given by the RGB values of A and B . This can only be selected for colors. The same value is output to all color components.	
DOT3_RGBA	Calculates the dot product of the vectors given by the RGB values of A and B . The color and alpha output values are the same.	

Note: The sum of **A** and **B** is clamped between 0.0 and 1.0 during ADD_MULT.

When using DOT3_RGBA, you must specify DOT3_RGBA for both the color and alpha values.

The following equation is used to calculate both DOT3_RGB and DOT3_RGBA.

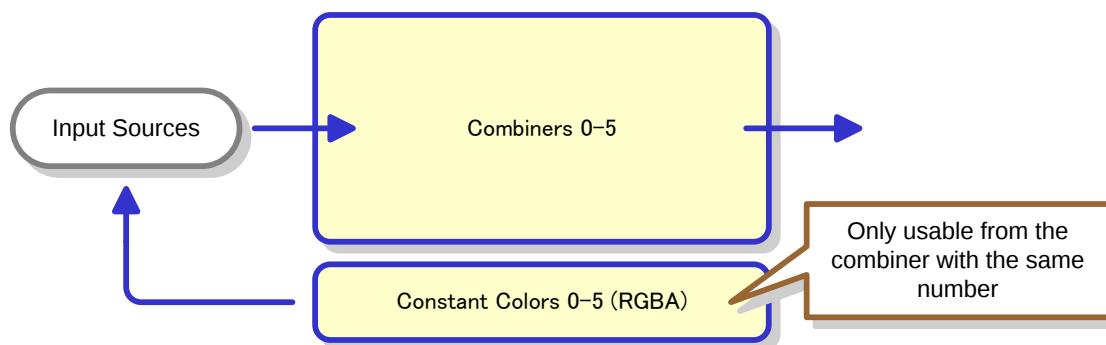
6.3.4 Scale

The *scale* is a factor by which to multiply the color and alpha values that have been blended by a combiner function. You can set a color scale and an alpha scale. You can choose a scale of x1, x2, or x4.

6.3.5 Constant Colors

A single constant color can be set for each texture combiner stage and used as an input source.

Figure 6-4 Constant Colors

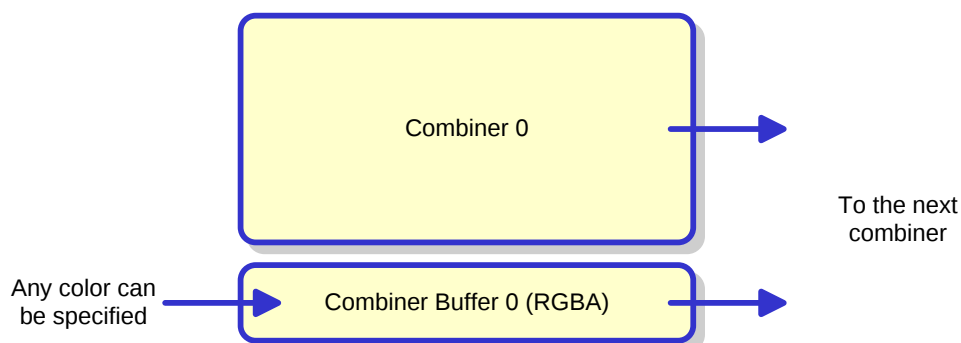


6.3.6 Combiner Buffers

A combiner buffer is used to temporarily store texture combiner output. Using a combiner buffer, you can use the output from the previous texture combiner stage as input to the next texture combiner stage.

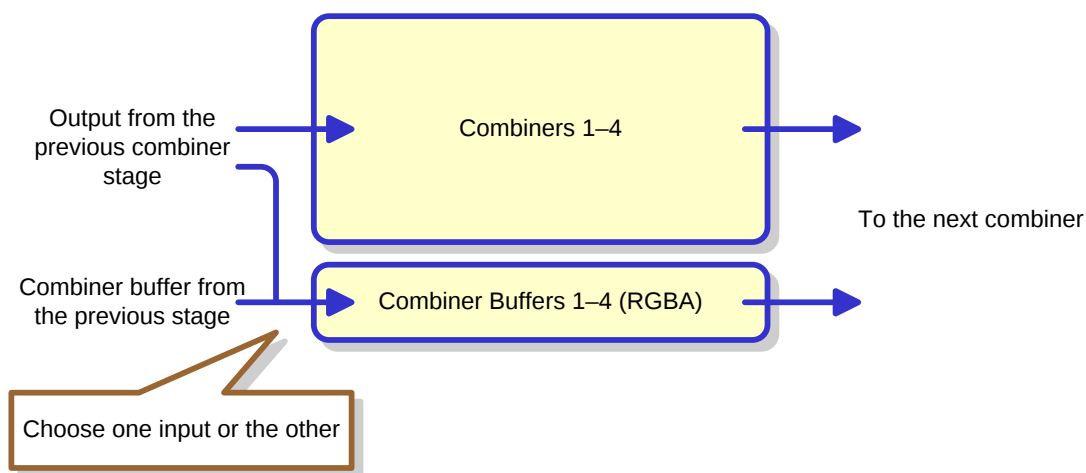
Like a constant color, combiner buffer 0 can be set equal to any color.

Figure 6-5 Combiner Buffer 0



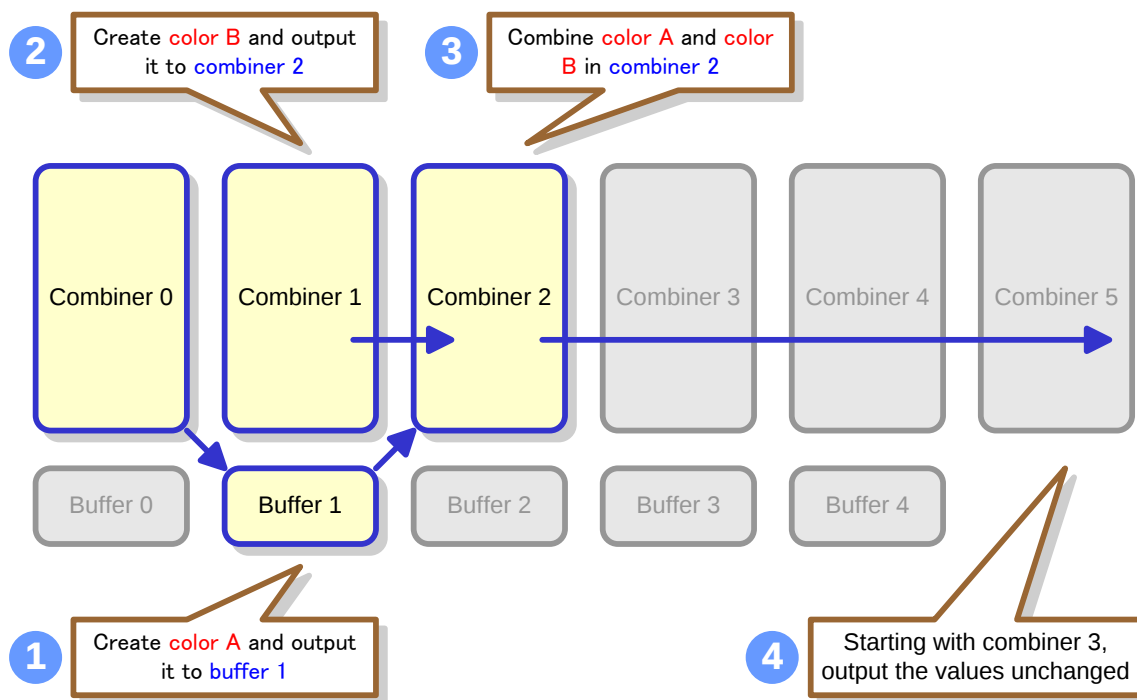
You can select either the output from the previous texture combiner or the color from the previous combiner buffer as the input to combiner buffers 1–4. You can choose both the color and alpha input.

Figure 6-6 Combiner Buffers 1–4



The following example shows how to use combiner buffers to create two types of colors and combine them in a later texture combiner stage.

Figure 6-7 Sample Use of Combiner Buffers

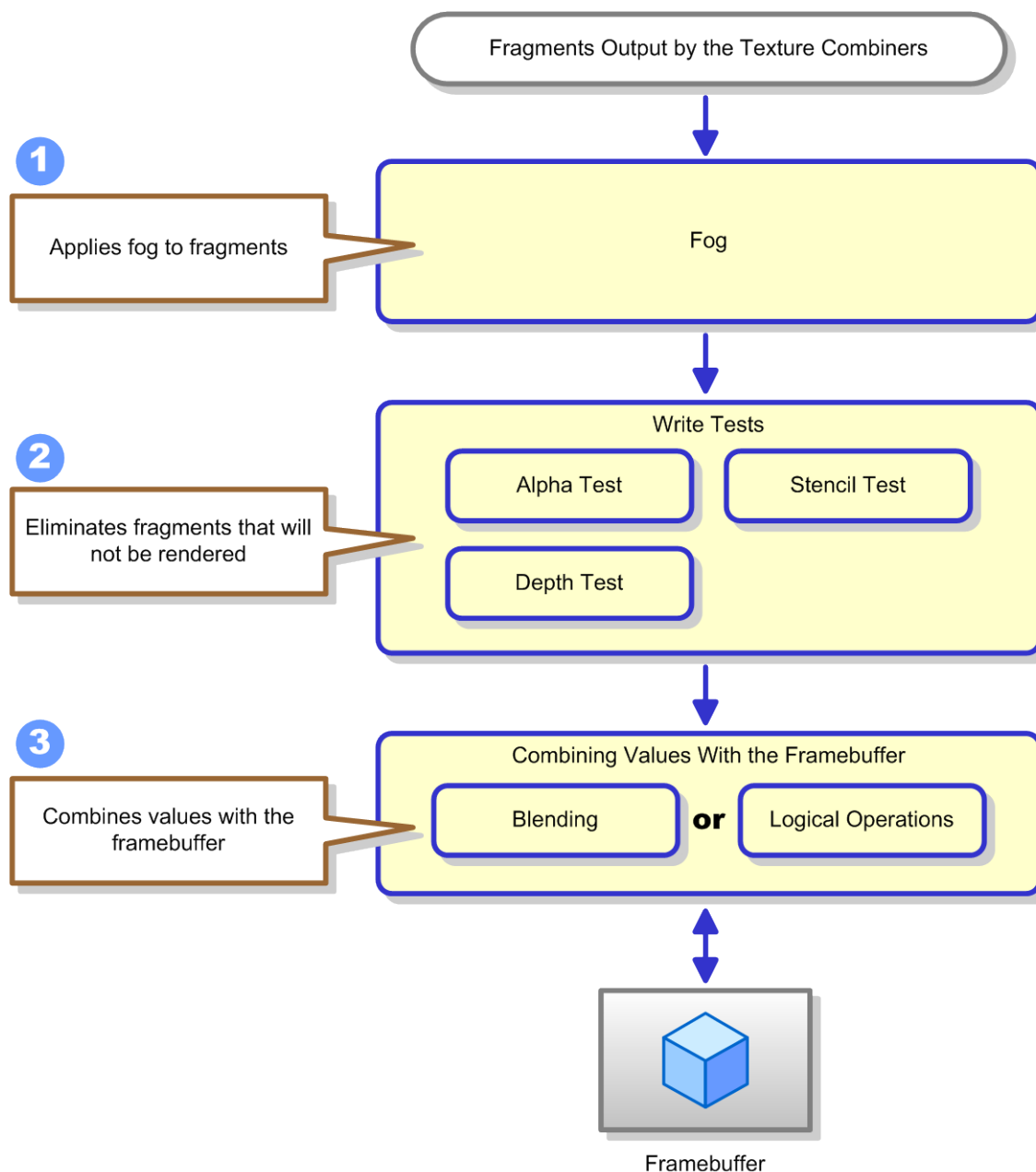


7 Fragment Operations

7.1 Overview of Fragment Operations

Fragment operations are run on fragments output by the texture combiners before they are written to the framebuffer (the buffer in which images are actually written). Fragment operations include fog, write tests, and value combinations with the framebuffer.

Figure 7-1 Overview of Fragment Operations



7.2 Fog

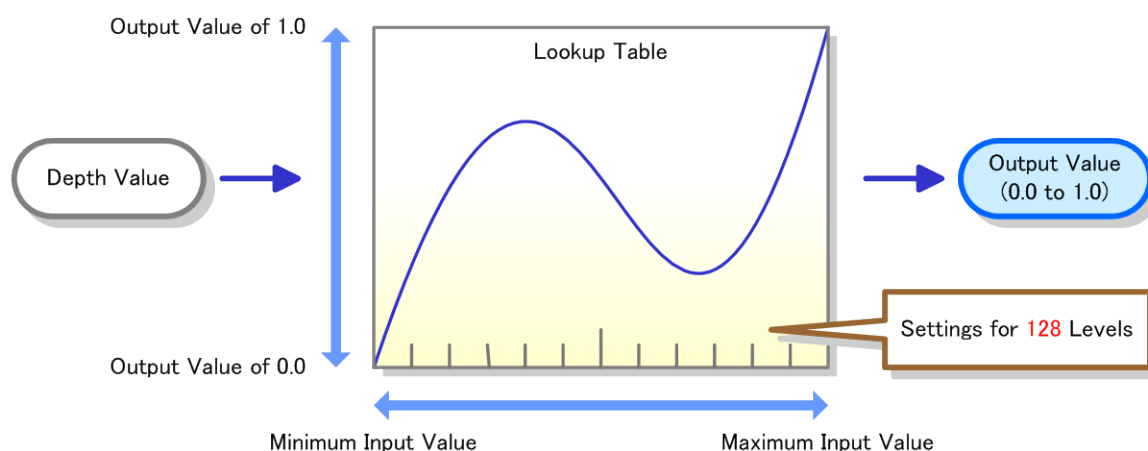
This process changes the color of a fragment based on its depth value (its apparent depth into the screen). Distant scenery can be made to appear misted over by fog. The following settings are available.

Table 7-1 Fog Settings

Setting	Description
Fog color	Sets the color to use for fog. Fragment colors are blended with the fog color.
Fog lookup table	A lookup table that configures how to apply fog by depth.

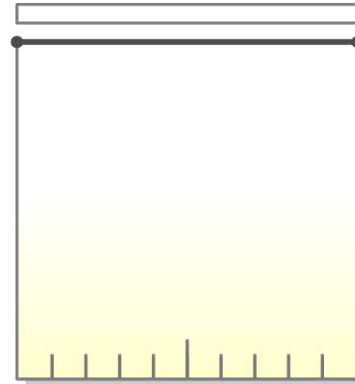
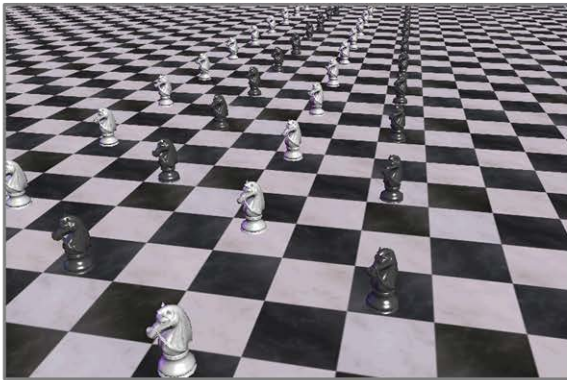
Fragment depth values are used as lookup table input. The lookup table is configured with output values for 128 levels of depth input values. Fog is calculated as follows.

Figure 7-2 Fog Lookup Table and Equation

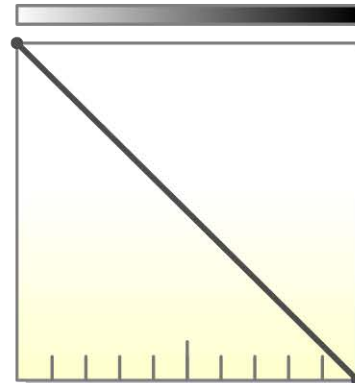


Fog Calculations

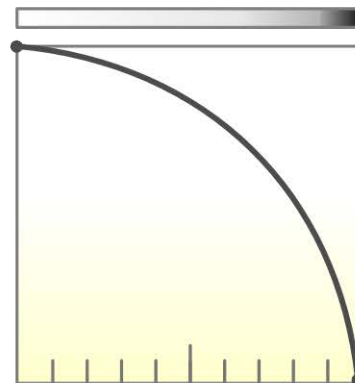
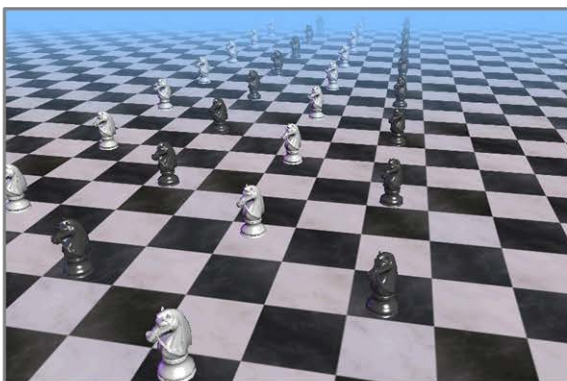
$$\text{Fragment Color} \times \text{Lookup Table Output} + \text{Fog Color} \times \left(1.0 - \text{Lookup Table Output} \right)$$

Figure 7-3 Fog**Example 1** No fog

Lookup Table

Example 2 Fog applied in proportion to distance

Lookup Table

Example 3 Fog increasing on a curve as depth increases

Lookup Table

7.3 Write Tests

By comparing fragments with lookup values and determining whether to actually write them to the framebuffer, unnecessary fragments are excluded from rendering. There are three types of tests, as follows.

Table 7-2 Write Tests

Type of Test	Description
Alpha test	Compares alpha values.
Depth test	Compares depth values.
Stencil test	Compares stencil values.

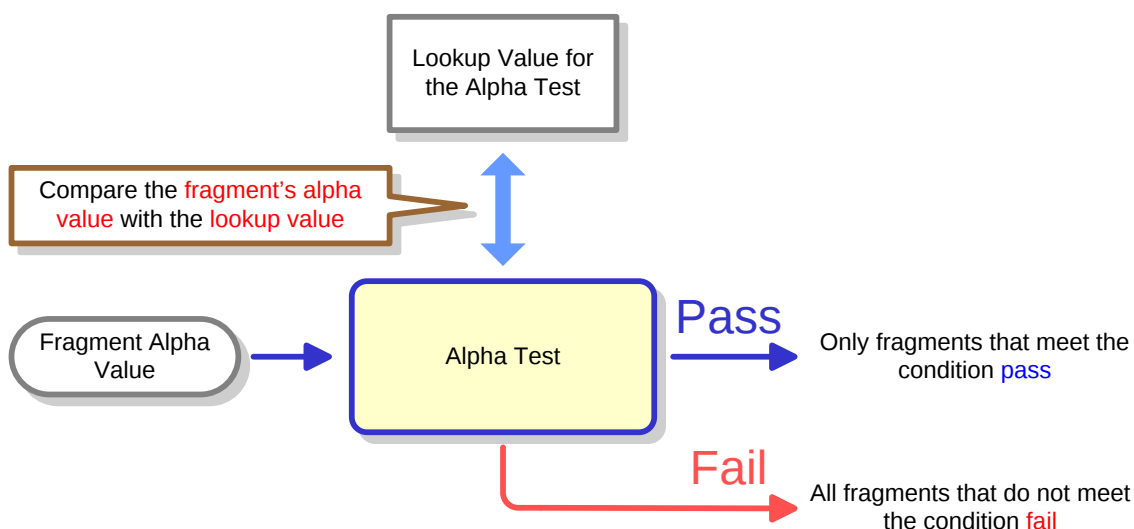
Note: The alpha test, stencil test, and depth test are run in that order.

The processing load increases when either the depth test or stencil test are used. The processing load is the same when both depth and stencil tests enabled and when just one or the other is enabled.

7.3.1 Alpha Test

Compares a fragment's alpha value with the lookup value for the alpha test. If the condition is not met, the fragment is thrown away.

Figure 7-4 Alpha Test



The alpha test uses the following two settings.

Table 7-3 Alpha Test Settings

Setting	Description
Lookup value	A value that can be freely set and is compared with a fragment's alpha value.
Comparison function	A function that compares a fragment's alpha value with a lookup value.

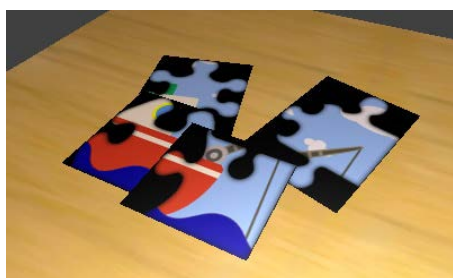
You can choose from the following eight types of comparison functions.

Table 7-4 Comparison Functions for the Alpha Test

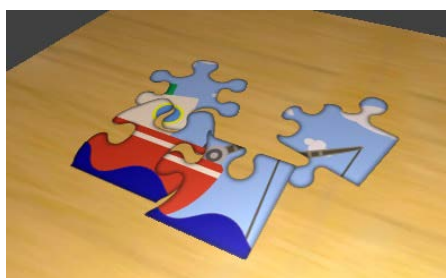
Comparison Function	Description
ALWAYS	Always passes.
NEVER	Never passes.
LESS	Passes if the fragment's alpha value is less than the lookup value.
LEQUAL	Passes if the fragment's alpha value is less than or equal to the lookup value.
GREATER	Passes if the fragment's alpha value is greater than the lookup value.
GEQUAL	Passes if the fragment's alpha value is greater than or equal to the lookup value.
EQUAL	Passes if the fragment's alpha value is equal to the lookup value.
NOTEQUAL	Passes if the fragment's alpha value is not equal to the lookup value.

By changing the alpha values of unnecessary texture regions and running the alpha test, you can prevent these unnecessary areas from being rendered.

Figure 7-5 Sample Usage of the Alpha Test



Without the Alpha Test

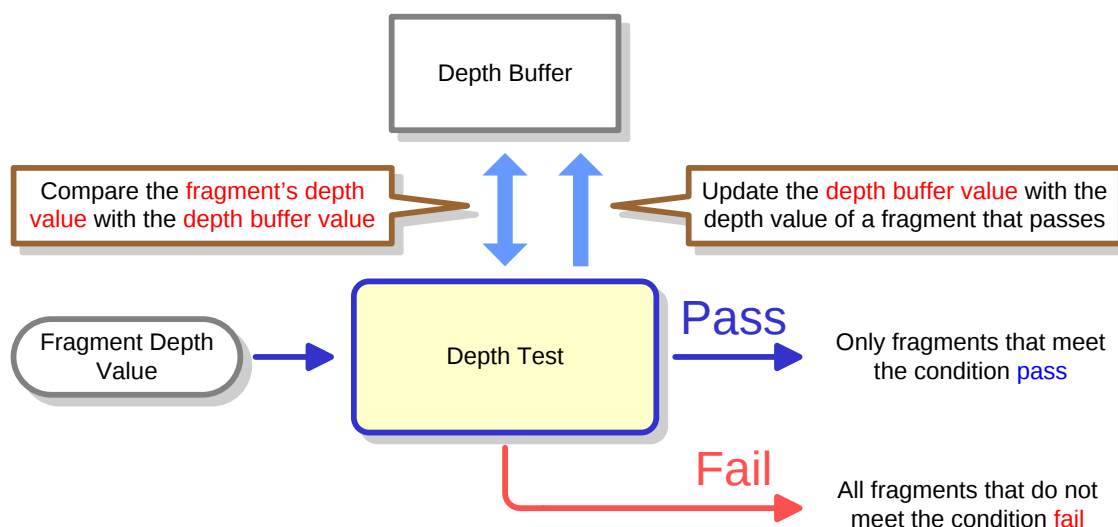


With the Alpha Test

7.3.2 Depth Test

Compares a fragment's depth value with a depth value from the depth buffer (which holds depth values for the framebuffer). If the condition is not met, the fragment is thrown away. The depth buffer can be updated with the depth value of a fragment that passes the test.

Figure 7-6 Depth Test

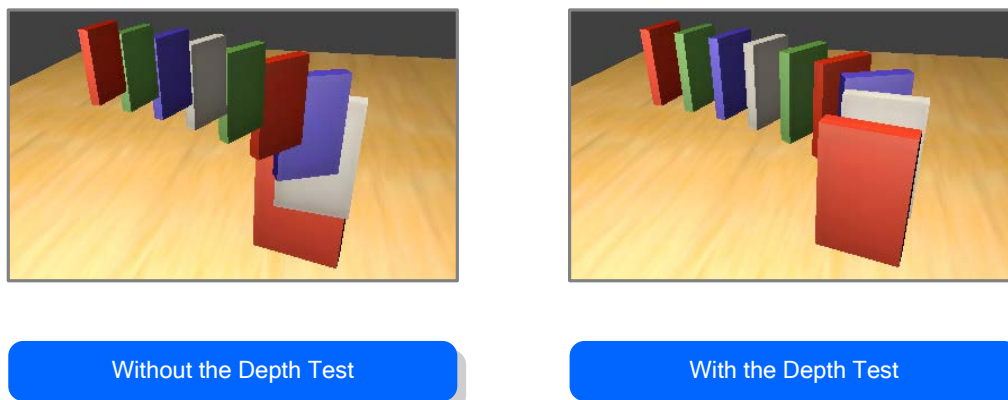


You can choose from the following eight types of comparison functions.

Table 7-5 Comparison Functions for the Depth Test

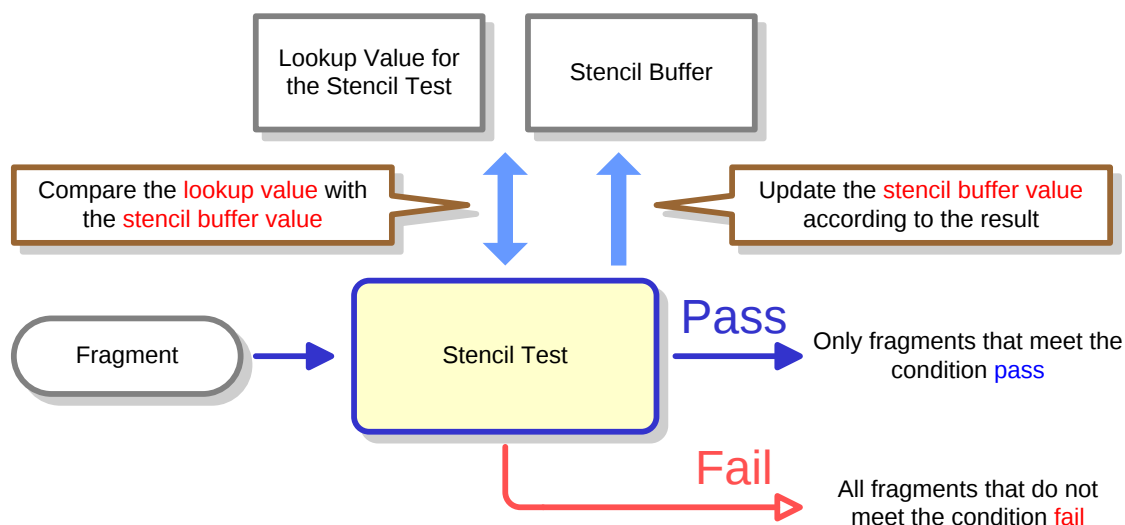
Comparison Function	Description
ALWAYS	Always passes.
NEVER	Never passes.
LESS	Passes if the fragment's depth value is less than the value in the depth buffer.
LEQUAL	Passes if the fragment's depth value is less than or equal to the value in the depth buffer.
GREATER	Passes if the fragment's depth value is greater than the value in the depth buffer.
GEQUAL	Passes if the fragment's depth value is greater than or equal to the value in the depth buffer.
EQUAL	Passes if the fragment's depth value is equal to the value in the depth buffer.
NOTEQUAL	Passes if the fragment's depth value is not equal to the value in the depth buffer.

Using the depth test, you can render objects so that they appear correctly front-to-back, regardless of the order in which they were actually rendered.

Figure 7-7 Sample Usage of the Depth Test

7.3.3 Stencil Test

Compares the lookup value for the stencil test with the value in the stencil buffer. If the condition is not met, the fragment is thrown away. Stencil buffer values can be updated according to the comparison results.

Figure 7-8 Stencil Test

The following two settings are available.

Table 7-6 Stencil Test Settings

Setting	Description
Lookup value	A value that can be freely set and is compared with the stencil buffer.
Comparison function	A function that compares the lookup value with the value in the stencil buffer.

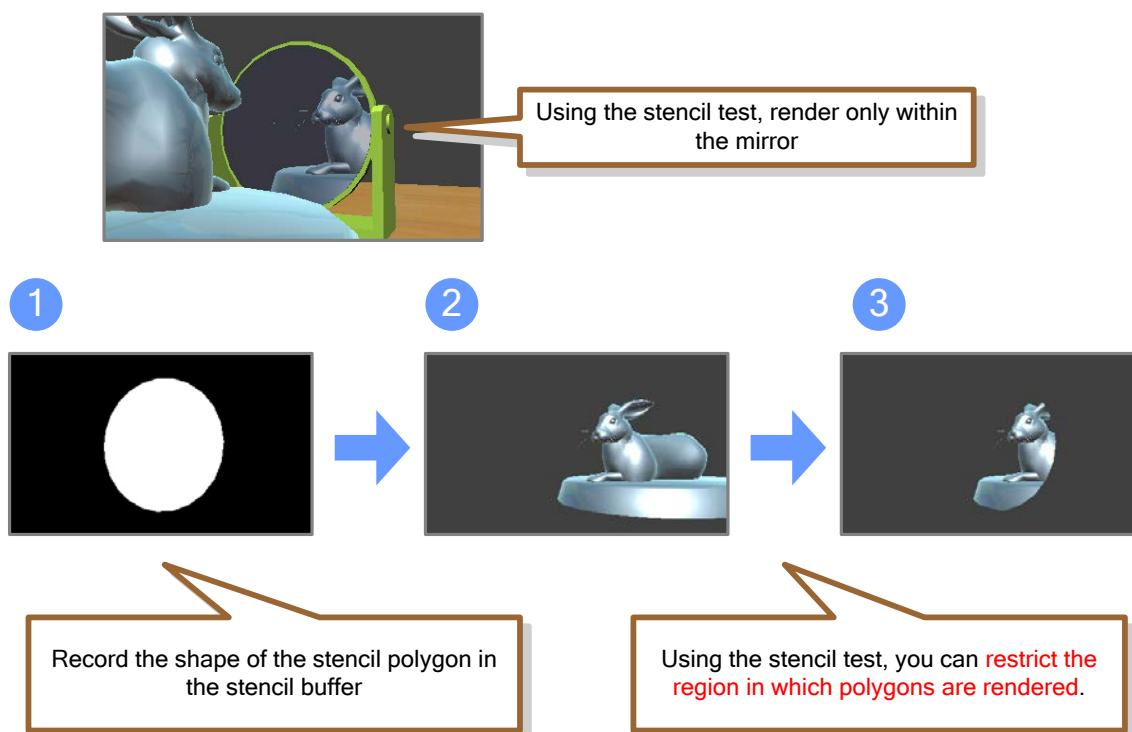
You can choose from the following eight types of comparison functions.

Table 7-7 Comparison Functions for the Stencil Test

Comparison Function	Description
ALWAYS	Always passes.
NEVER	Never passes.
LESS	Passes if the lookup value is less than the value in the stencil buffer.
LEQUAL	Passes if the lookup value is less than or equal to the value in the stencil buffer.
GREATER	Passes if the lookup value is greater than the value in the stencil buffer.
GEQUAL	Passes if the lookup value is greater than or equal to the value in the stencil buffer.
EQUAL	Passes if the lookup value is equal to the value in the stencil buffer.
NOTEQUAL	Passes if the lookup value is not equal to the value in the stencil buffer.

As shown in the following figure, the stencil test is normally used to render only within a masked region that has an irregular shape.

Figure 7-9 Sample Usage of the Stencil Test



7.4 Writing to the Framebuffer

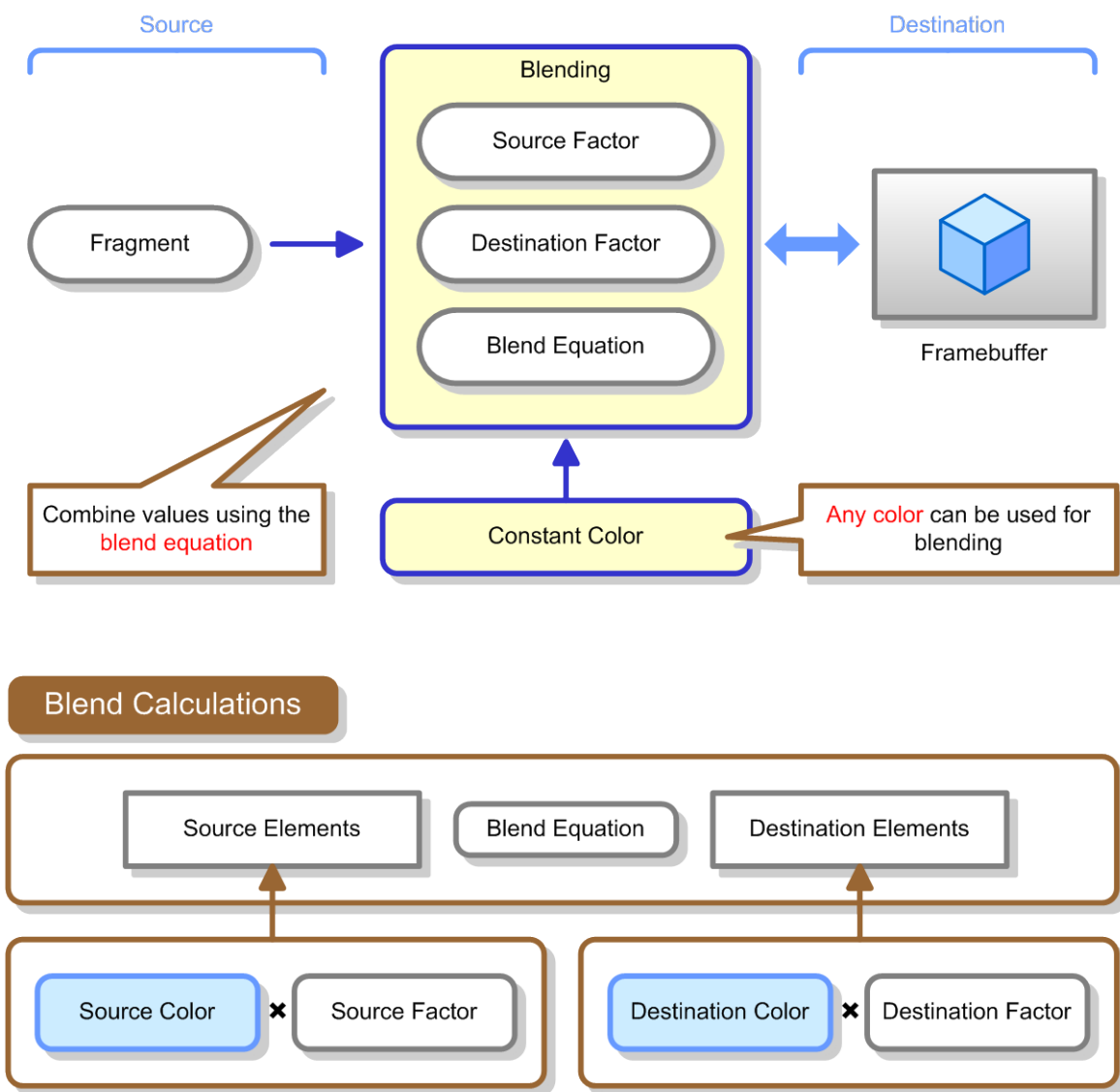
If a fragment passes the write test, it is written to the framebuffer. Values are combined with (written to) the framebuffer in one of the following two ways.

- Blending
- Logical operations

7.4.1 Blending

Blending combines the new fragment color to write (the source) with the framebuffer color being written to (the destination) through multiplication by factors. If the framebuffer supports alpha values, they are blended as well.

Figure 7-10 Blending



Blending uses the following four settings.

Table 7-8 Blend Settings

Setting	Description
Source factor	The factor by which to multiply the source.
Destination factor	The factor by which to multiply the destination.
Constant color	An arbitrary (RGBA) color used by the source and destination factors.
Blending equation	The equation to use to blend the source and destination.

You can choose a source and a destination factor from the following 15 types for both the color and the alpha values. The table uses **S**, **D**, and **C** to represent the source, destination, and constant color, respectively. The subscripts **r**, **g**, **b**, and **a** indicate the components.

Table 7-9 Source and Destination Factors

Factor	RGB Factor (R, G, B)	Alpha Factor (A)
ZERO	(0, 0, 0)	0
ONE	(1, 1, 1)	1
SRC_COLOR	(Sr , Sg , Sb)	Sa
1 - SRC_COLOR	(1 - Sr , 1 - Sg , 1 - Sb)	1 - Sa
DST_COLOR	(Dr , Dg , Db)	Da
1 - DST_COLOR	(1 - Dr , 1 - Dg , 1 - Db)	1 - Da
SRC_ALPHA	(Sa , Sa , Sa)	Sa
1 - SRC_ALPHA	(1 - Sa , 1 - Sa , 1 - Sa)	1 - Sa
DST_ALPHA	(Da , Da , Da)	Da
1 - DST_ALPHA	(1 - Da , 1 - Da , 1 - Da)	1 - Da
CONSTANT_COLOR	(Cr , Cg , Cb)	Ca
1 - CONSTANT_COLOR	(1 - Cr , 1 - Cg , 1 - Cb)	1 - Ca
CONSTANT_ALPHA	(Ca , Ca , Ca)	Ca
1 - CONSTANT_ALPHA	(1 - Ca , 1 - Ca , 1 - Ca)	1 - Ca
SRC_ALPHA_SATURATE	Compares Sa with 1 - Da and then applies the smaller one to the RGB values.	1

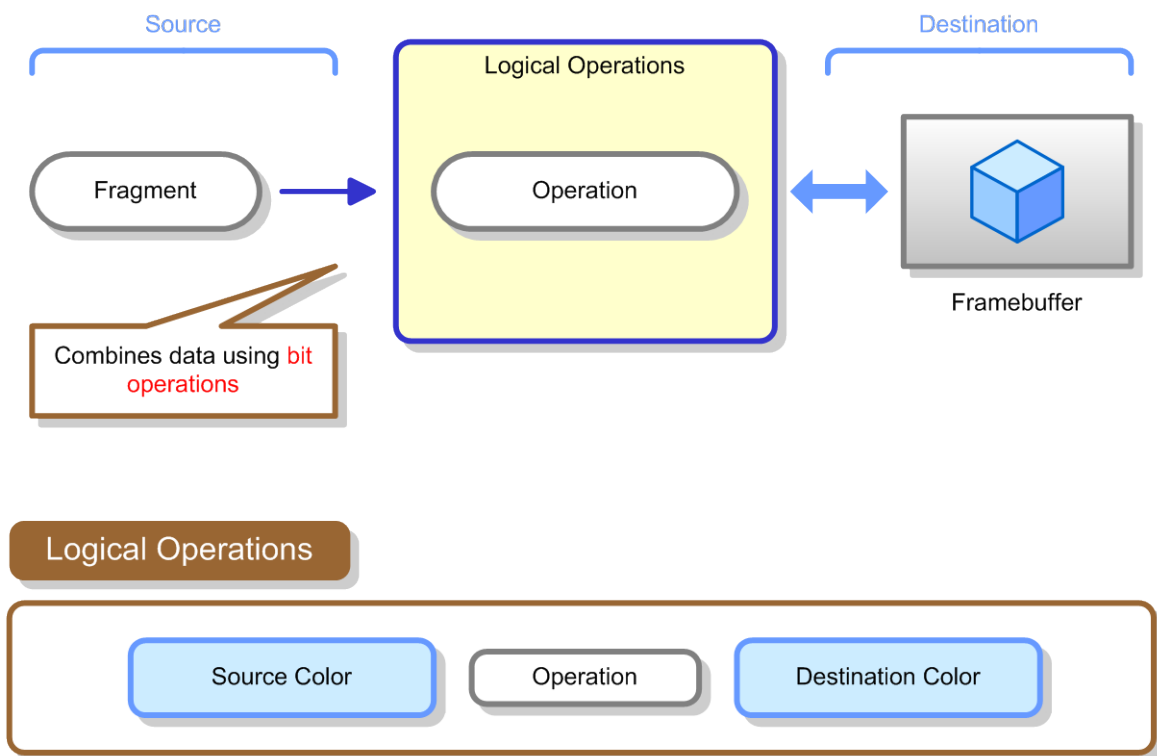
You can choose from the following five equations for both the color and the alpha values. The blending equations calculate results that are written to the framebuffer. The table uses **S** to represent the product of the source and the source factor and **D** to represent the product of the destination and the destination factor.

Table 7-10 Blending Equations

Blending Equation	Calculation
ADD	$S + D$
SUBTRACT	$S - D$
REVERSE_SUBTRACT	$D - S$
MIN	Compares S and D and then uses the smaller value.
MAX	Compares S and D and then uses the larger value.

7.4.2 Logical Operations

Logical operations are run on the source and destination and then the results are written to the framebuffer.

Figure 7-11 Logical Operations

You can choose from the following 16 operations.

Table 7-11 Logical Operations

Method	Operation	Description
CLEAR	0	Writes 0 to the RGBA values.
COPY	s	Writes the source unchanged.
NOOP	d	Does not write anything.
SET	1	Writes 1 to the RGBA values.
COPY_INVERTED	$\neg s$	Inverts the source.
INVERT	$\neg d$	Negates the destination.
AND_REVERSE	$s \wedge \neg d$	The logical AND of the source and the negated destination.
OR_REVERSE	$s \vee \neg d$	The logical OR of the source and the inverted destination.
AND	$s \wedge d$	The logical AND of the source and destination.
OR	$s \vee d$	The logical OR of the source and destination.
NAND	$\neg (s \wedge d)$	The negated logical AND of the source and destination.
NOR	$\neg (s \vee d)$	The negated logical OR of the source and destination.
XOR	$s \text{ XOR } d$	The exclusive OR of the source and destination.
EQUIV	$\neg (s \text{ XOR } d)$	The negated exclusive OR of the source and destination.
AND_INVERTED	$\neg s \wedge d$	The logical AND of the negated source and the destination.
OR_INVERTED	$\neg s \vee d$	The logical OR of the negated source and the destination.

8 Framebuffers

8.1 Framebuffer Formats

Rendering results are written to the framebuffer. The framebuffer comprises the following three types of buffers.

Table 8-1 Framebuffer Structure

Buffer	Purpose
Color buffer	A buffer used to render RGB and RGBA images.
Depth buffer	A buffer that stores the depth values used by the depth test.
Stencil buffer	A buffer that stores the stencil values used by the stencil test.

These buffers have the following formats. The number of bits per pixel varies by format. A format with fewer bits per pixel loses as much accuracy as it saves on the buffer size allocated in memory.

Table 8-2 Buffer Format

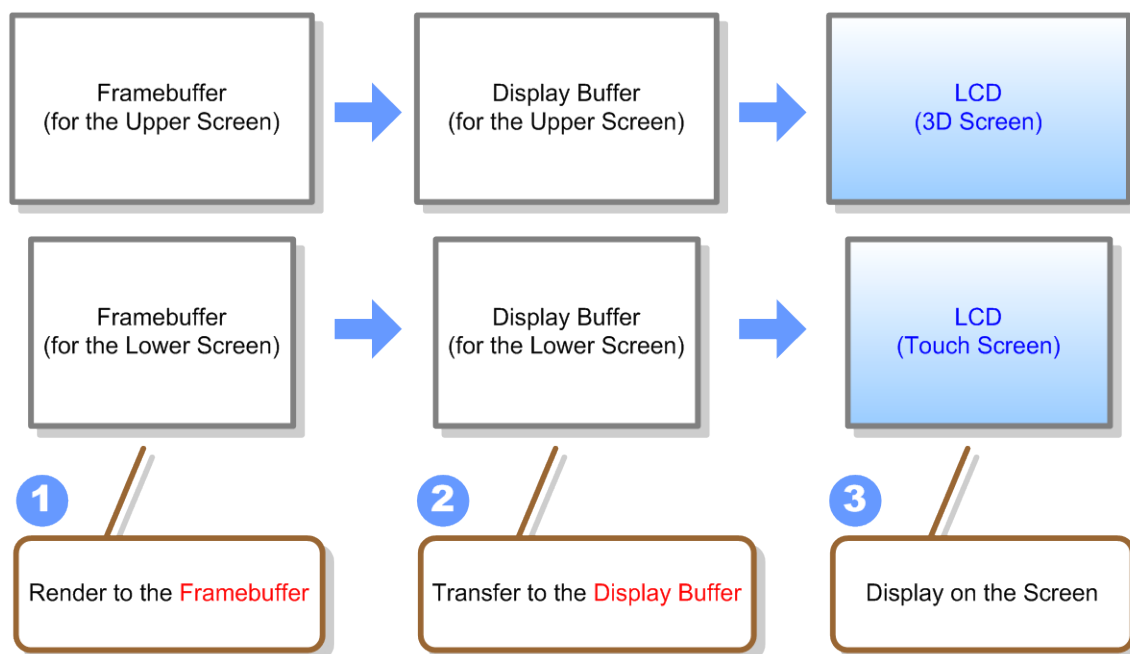
Buffer	Format	Internal Structure (Bits)
Color buffer	RGB565	R5 G6 B5
	RGBA4	R4 G4 B4 A4
	RGB5A1	R5 G5 B5 A1
	RGBA8	R8 G8 B8 A8
Depth buffer	DEPTH16	16
	DEPTH24	24
Stencil buffer	STENCIL8	8

Note: You must set DEPTH24 as the depth buffer format when you use the stencil buffer.

8.2 LCD Output

Rendering results in the framebuffer are transferred to a display buffer and then ultimately output to the LCDs.

Figure 8-1 LCD Output



8.3 Antialiasing

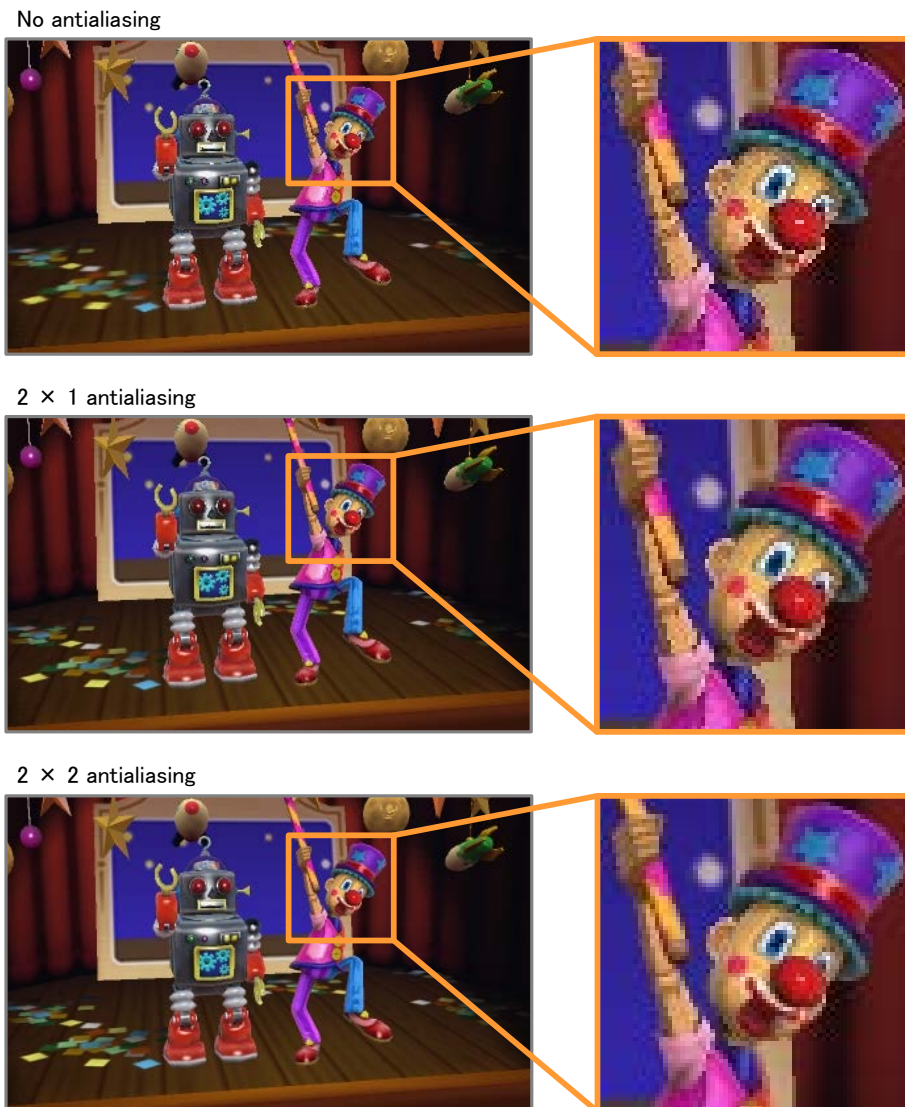
Antialiasing is a technique that smooths polygon edges.

Antialiasing can be applied by rendering the framebuffer at a higher resolution than the display buffer and then shrinking images when they are transferred from the framebuffer to the display buffer. To shrink images, the size of the original framebuffer must be some multiple of the display buffer.

The following two antialiasing methods are possible.

Table 8-3 Antialiasing

How to Shrink	Antialiasing Direction	Required Framebuffer Size
2x1	Only vertically on the LCDs	Twice the display buffer
2x2	Vertically and horizontally on the LCDs	Four times the display buffer

Figure 8-2 Antialiasing

8.4 Application to Textures

Rendering results can be used as textures. The following two methods are available.

Table 8-4 Application to Textures

How to Apply to Textures	Description
Copy-to-texture	Copies the content of the framebuffer into texture memory.
Render-to-texture	Renders directly to texture memory. This is faster than copy-to-texture.

9 Special Effects

9.1 Gas

(TBD)

9.2 Shadows

(TBD)

10 Samples

10.1 Sample Settings for Fragment Lighting

This chapter gives sample settings for fragment lighting, which is an important element of CTR graphics.

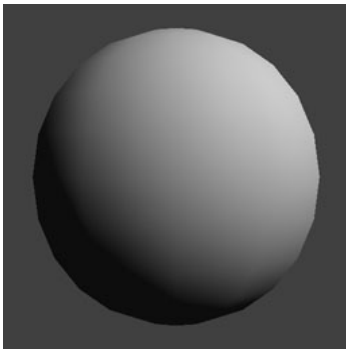
For details on each of the settings used by fragment lighting, see Chapter 4 Fragment Lighting.

Any color or lookup table output that is not explicitly set by an example is assumed to have a value of 0 (black) or 1, respectively. Texture combiners are assumed to be configured to add the primary and secondary colors.

10.1.1 Lambert

Lambert shading depicts ambient and diffuse colors. The ambient color is spread evenly over the entire object. The diffuse color determines the brightness based on the fragment and light positions; it is unrelated to the viewer's position.

Figure 10-1 Example of Lambert Shading



The ambient and diffuse terms are output to the primary color. Lambert shading uses the following settings.

Figure 10-2 Settings for the Lambert Shading Model

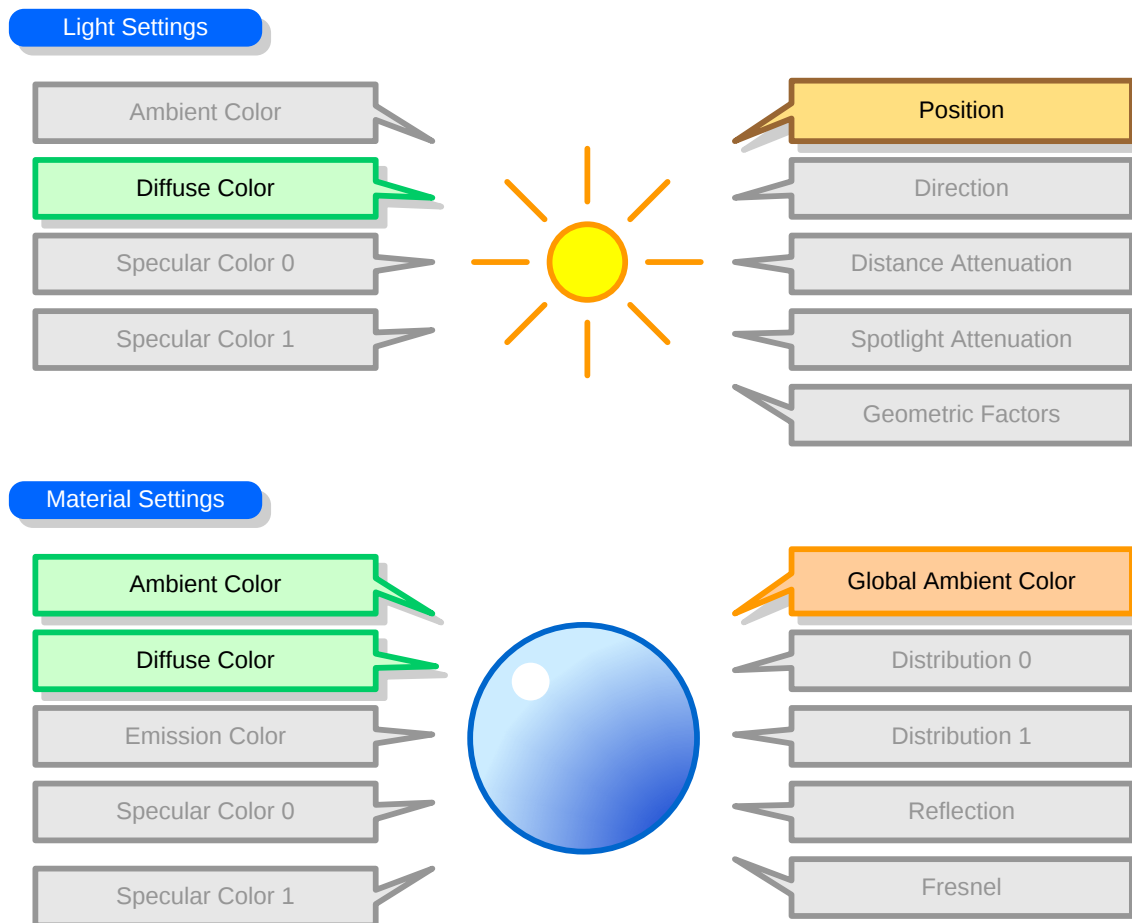
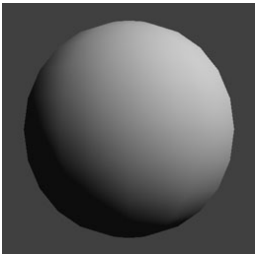
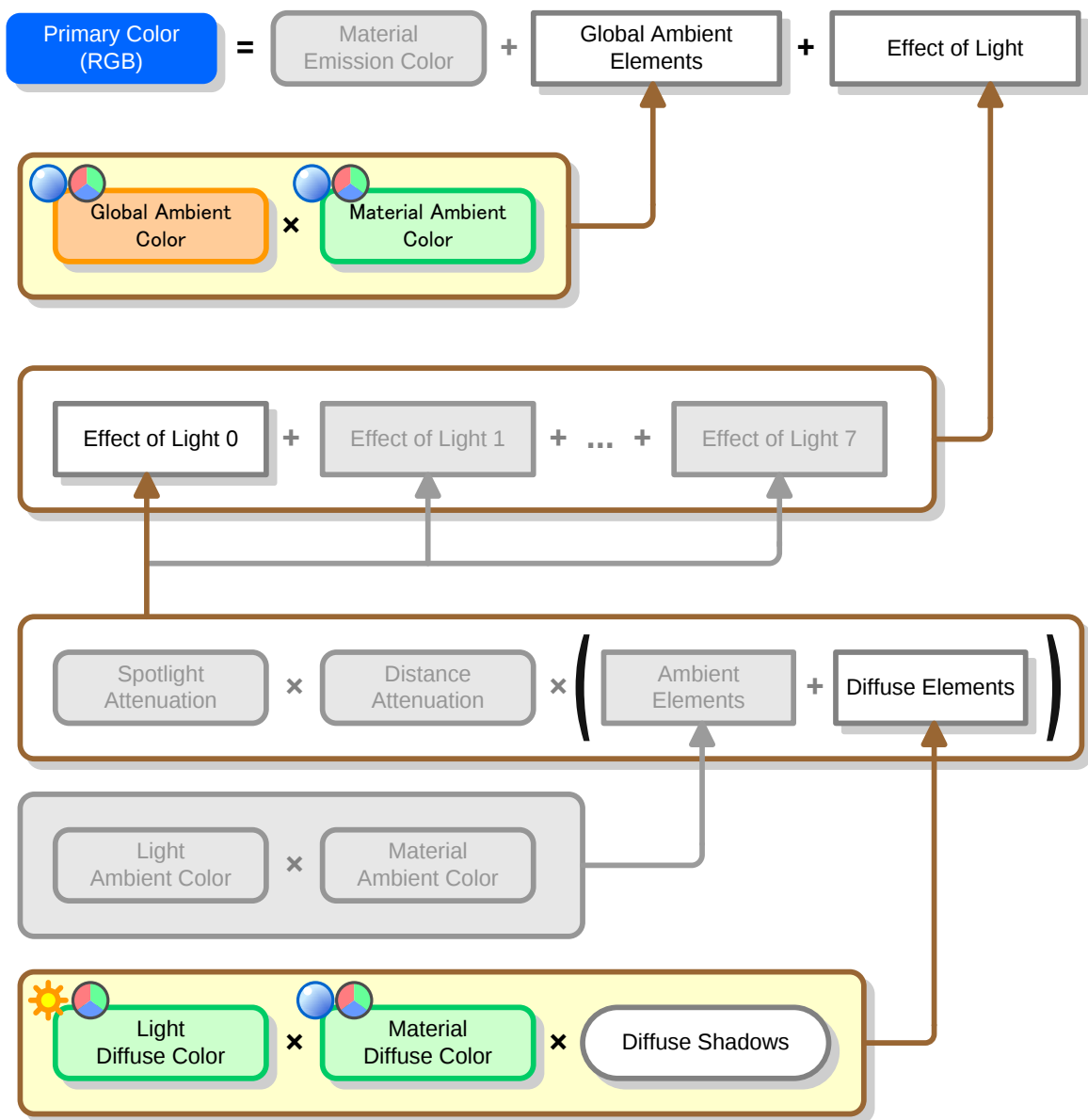


Table 10-1 Sample Settings for the Lambert Shading Model (Primary Color)

Units	Setting	Example	Lighting Result
Lights 	Diffuse Color		
	Position	Any coordinates	
Materials 	Ambient Color		
	Diffuse Color		
	Global Ambient Color		

Lambert shading is calculated using the following formula.

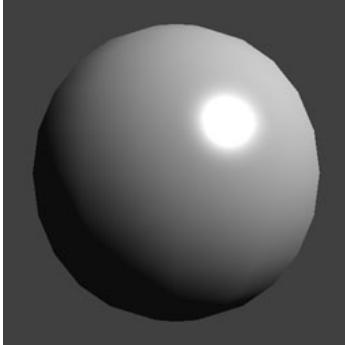
Figure 10-3 Formula for Lambert Shading (Primary Color)



10.1.2 Phong

Phong shading depicts specularity in addition to Lambert shading.

Figure 10-4 Example of Phong Shading



Lighting calculations use the distribution term to generate the specular color, which is output to the secondary color. Phong shading uses the following settings.

Figure 10-5 Settings for the Phong Shading Model

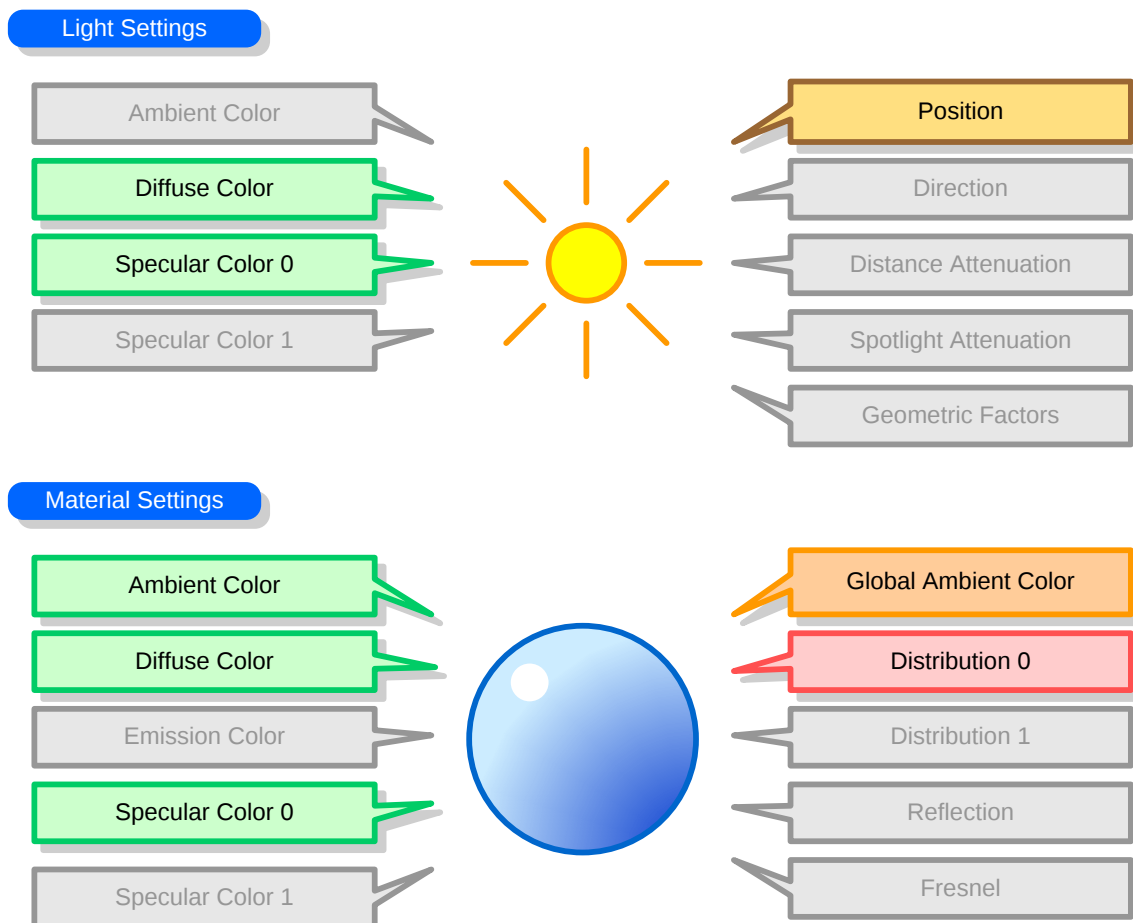


Table 10-2 Sample Settings for the Phong Shading Model (Primary Color)

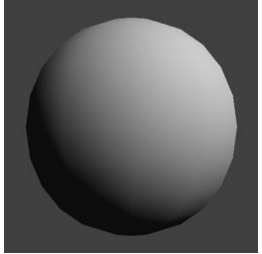
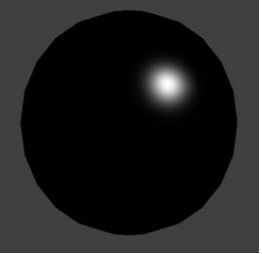
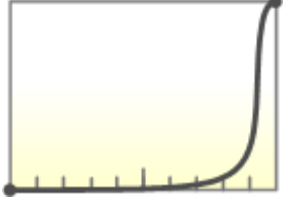
Units	Setting	Example	Lighting Result
Lights 	Diffuse Color		
	Position	Any coordinates	
Materials 	Ambient Color		
	Diffuse Color		
	Global Ambient Color		

Table 10-3 Sample Settings for the Phong Shading Model (Secondary Color)

Units	Setting	Example	Lighting Result
Lights 	Specular Color 0		
	Position	Any coordinates	
Materials 	Specular Color 0		
	Distribution 0 (Input)	NH	
	Distribution 0 (LUT)		

Phong shading is calculated using the following formulas.

Figure 10-6 Formula for Phong Shading (Primary Color)

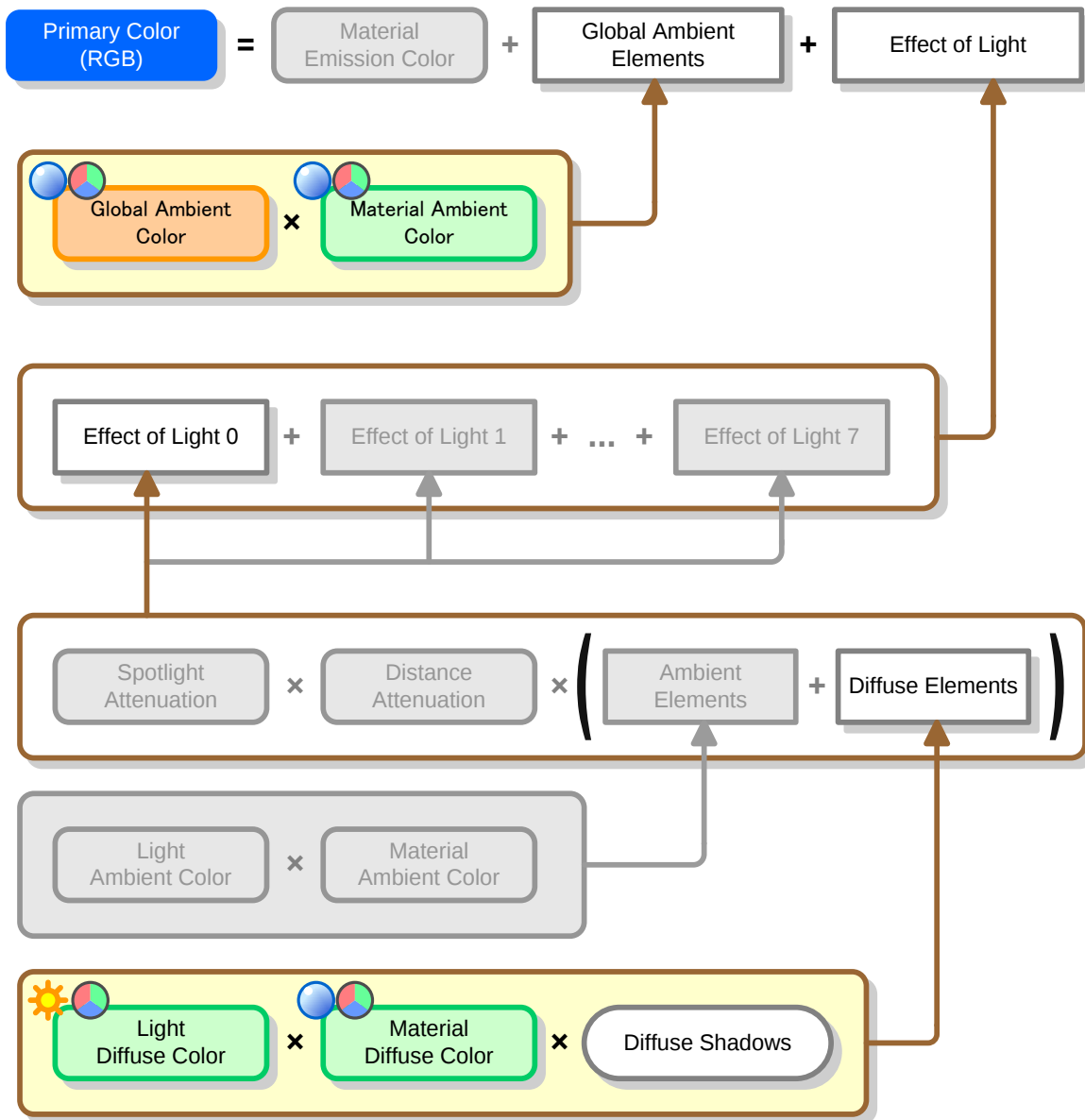
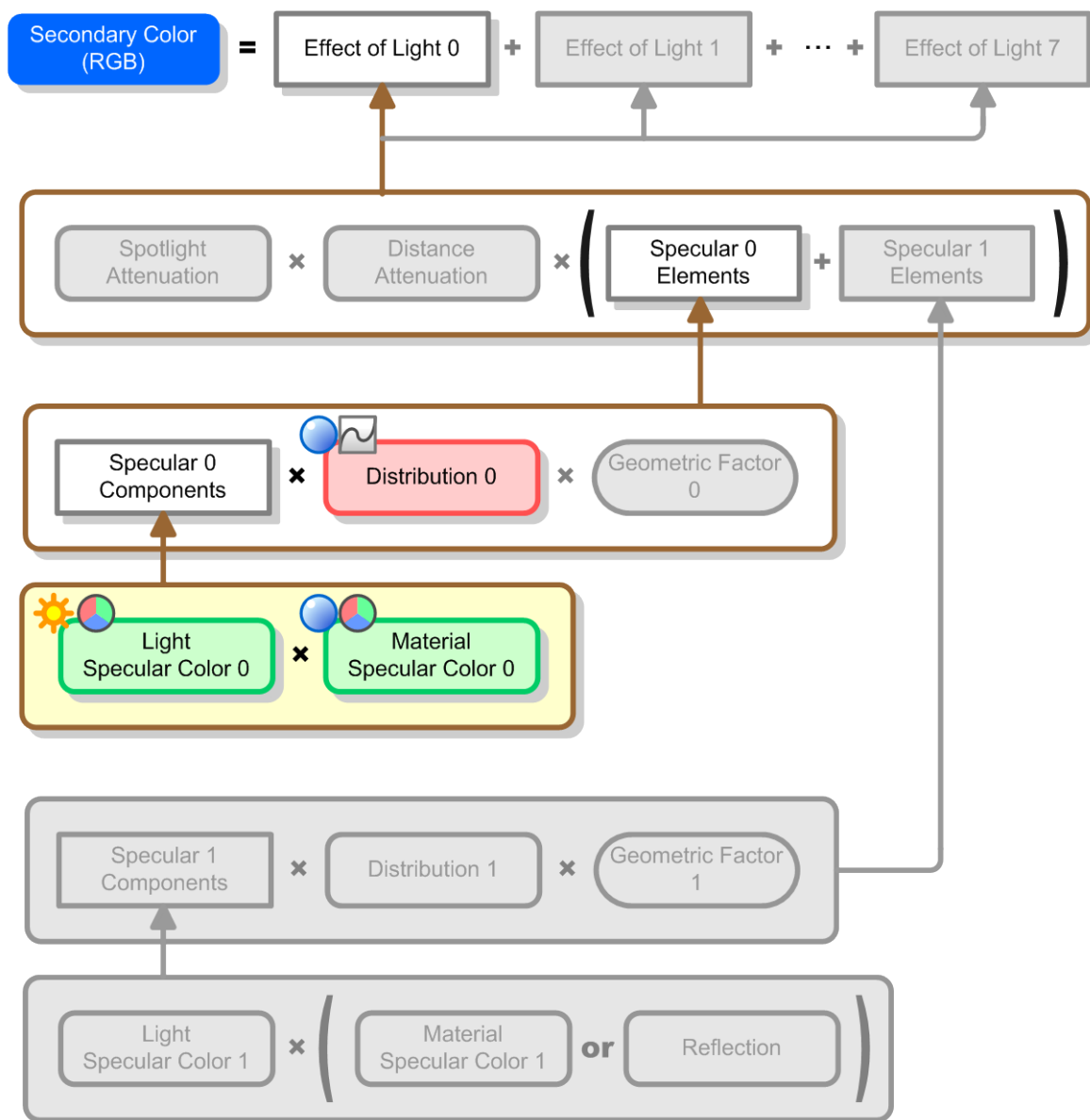


Figure 10-7 Formula for Phong Shading (Secondary Color)



10.1.3 Toon

Toon shading depicts shadows with distinct boundaries.

Figure 10-8 Example of Toon Shading



By using a tiered lookup table for its distribution term and treating the input value as LN, toon shading shows distinct lighting boundaries. Lighting results are output to the secondary color. Toon shading uses the following settings.

Figure 10-9 Settings for the Toon Shading Model

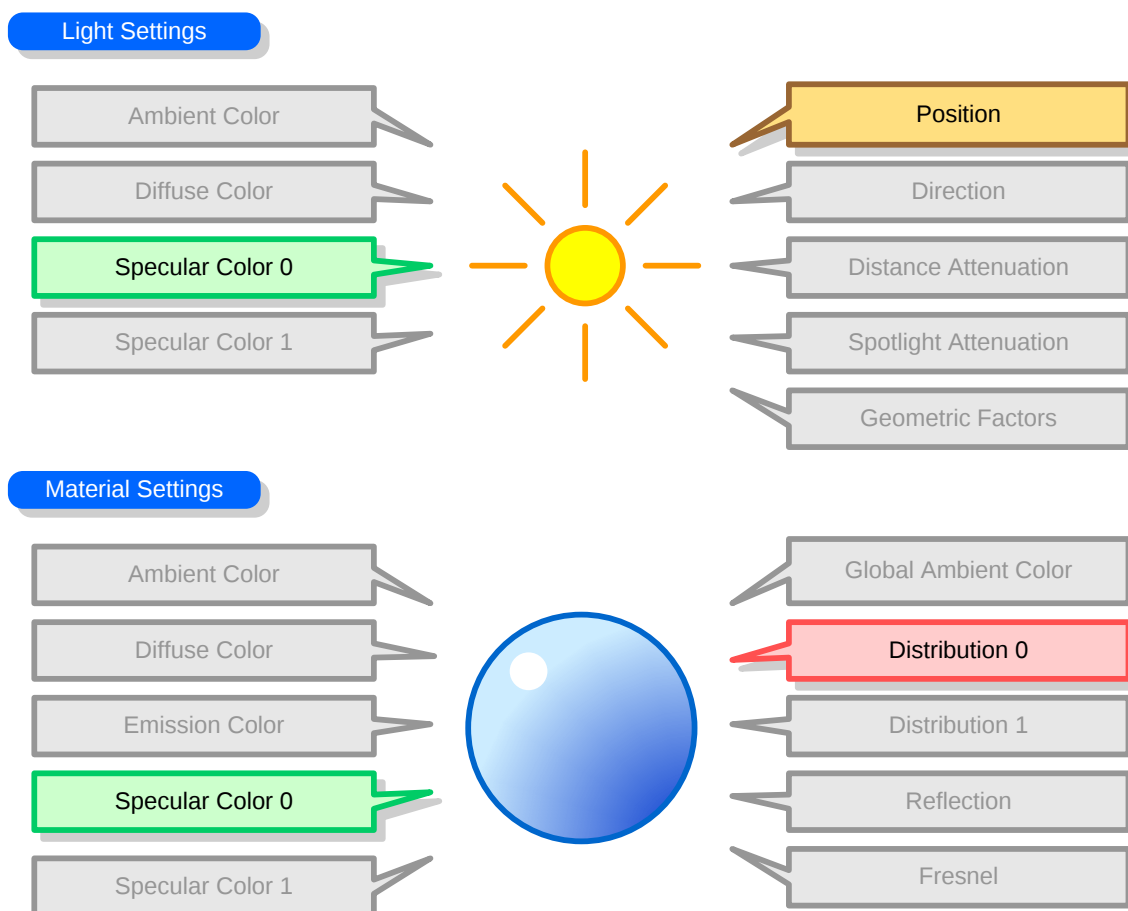
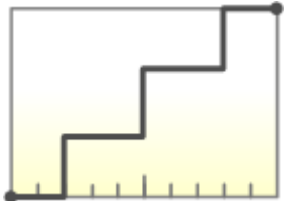
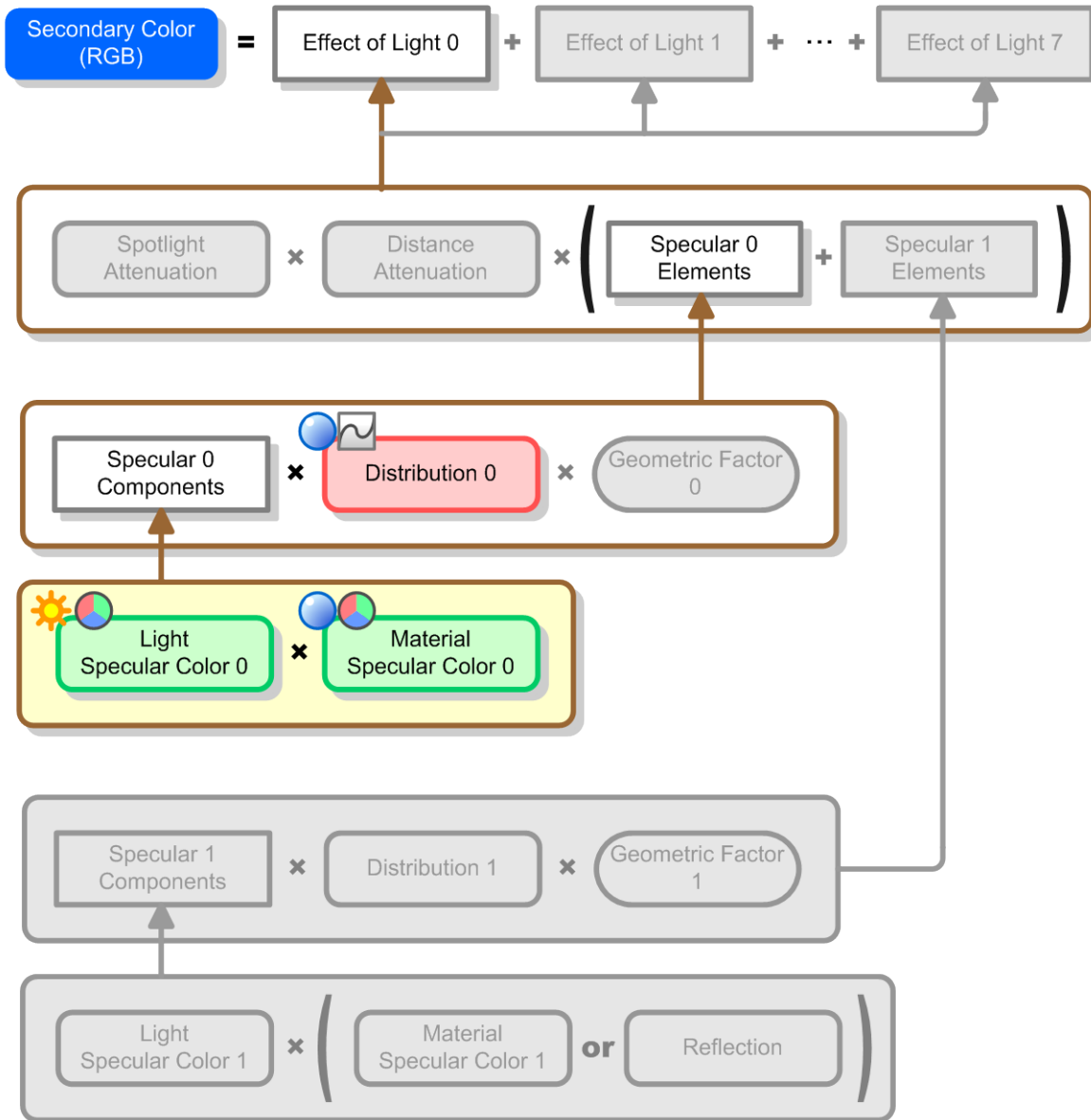


Table 10-4 Sample Settings for the Toon Shading Model (Secondary Color)

Units	Setting	Example	Lighting Result
Lights 	Specular Color 0		
	Position	Any coordinates	
Materials 	Specular Color 0		
	Distribution 0 (Input)	LN	
	Distribution 0 (LUT)		

Toon shading is calculated using the following formula.

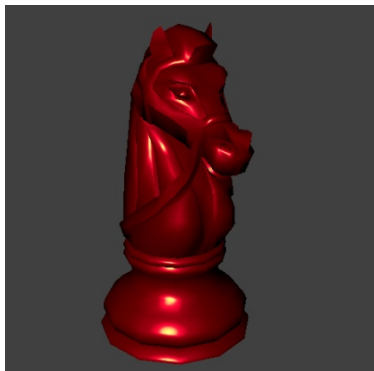
Figure 10-10 Formula for Toon Shading (Secondary Color)



10.1.4 Two-Layer Paint

The two-layer paint model depicts materials, such as a car body, that have two surface coatings.

Figure 10-11 Example of Two-Layer Paint



In addition to the diffuse color, the two-layer paint model sets differently-shaped lookup tables for distribution 0 and distribution 1 to represent the specularity of each paint layer. Specular lighting results are output to the secondary color.

Figure 10-12 Settings for the Two-Layer Paint Model

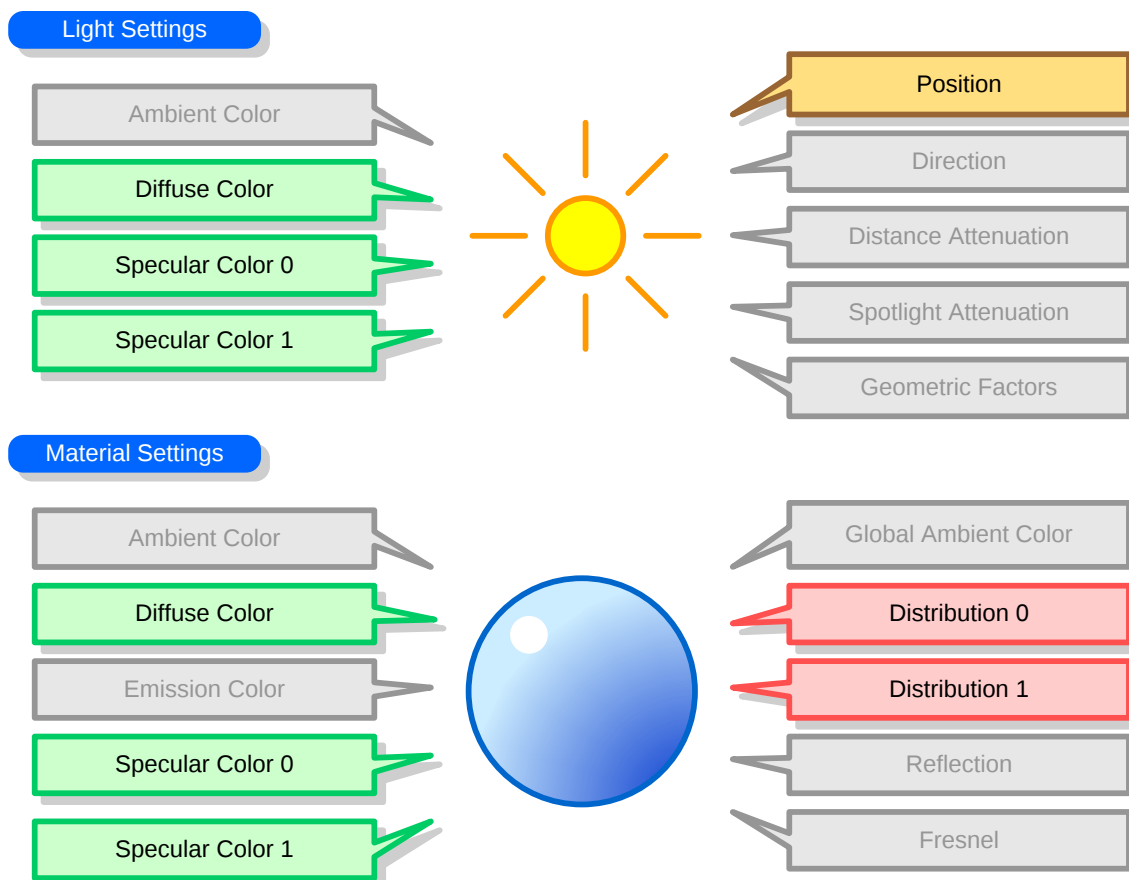


Table 10-5 Sample Settings for the Two-Layer Paint Model (Primary Color)

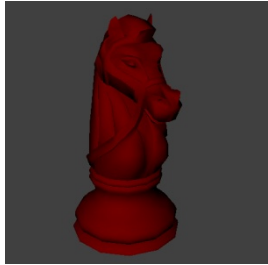
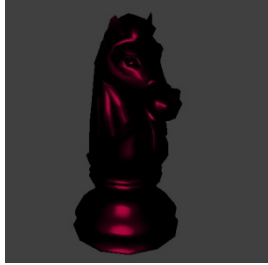
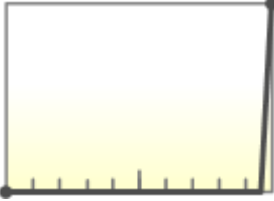
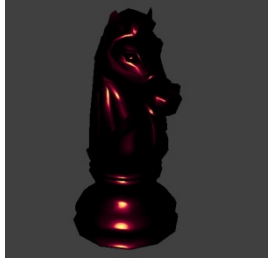
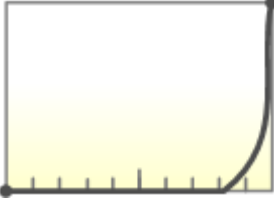
Units	Setting	Example	Lighting Result
Lights 	Diffuse Color		
	Position	Any coordinates	
Materials 	Diffuse Color		

Table 10-6 Sample Settings for the Two-Layer Paint Model (Secondary Color)

Units	Setting	Example	Lighting Result
Lights 	Specular Color 0		Specular 0 Element 
	Specular Color 1		
	Position	Any coordinates	
Materials 	Specular Color 0		Specular 1 Element 
	Distribution 0 (Input)	NH	
	Distribution 0 (LUT)		Final Secondary Color 
	Specular Color 1		
	Distribution 1 (Input)	NH	
	Distribution 1 (LUT)		

The two-layer paint model is calculated using the following formulas.

Figure 10-13 Formula for Two-Layer Paint (Primary Color)

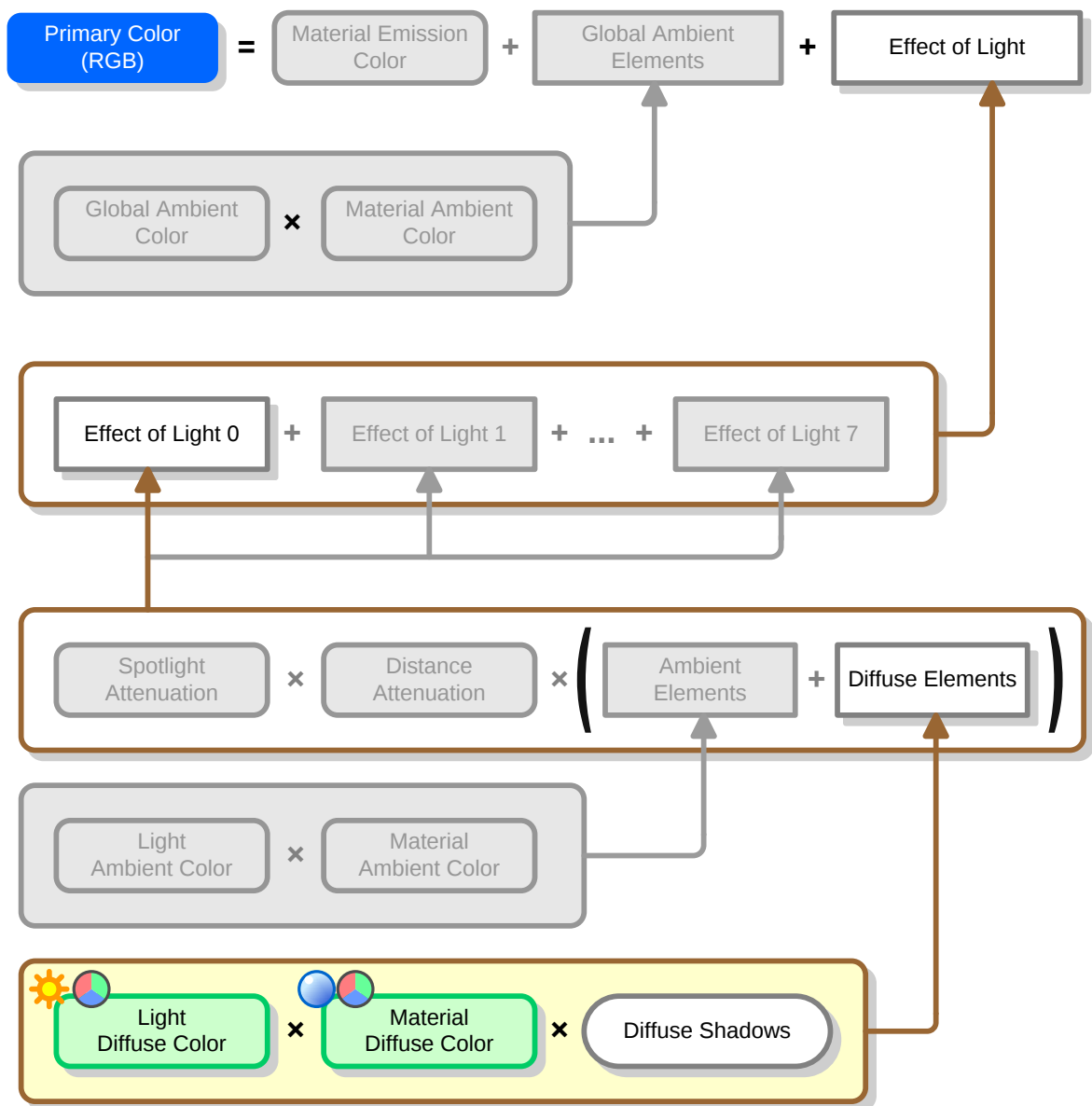
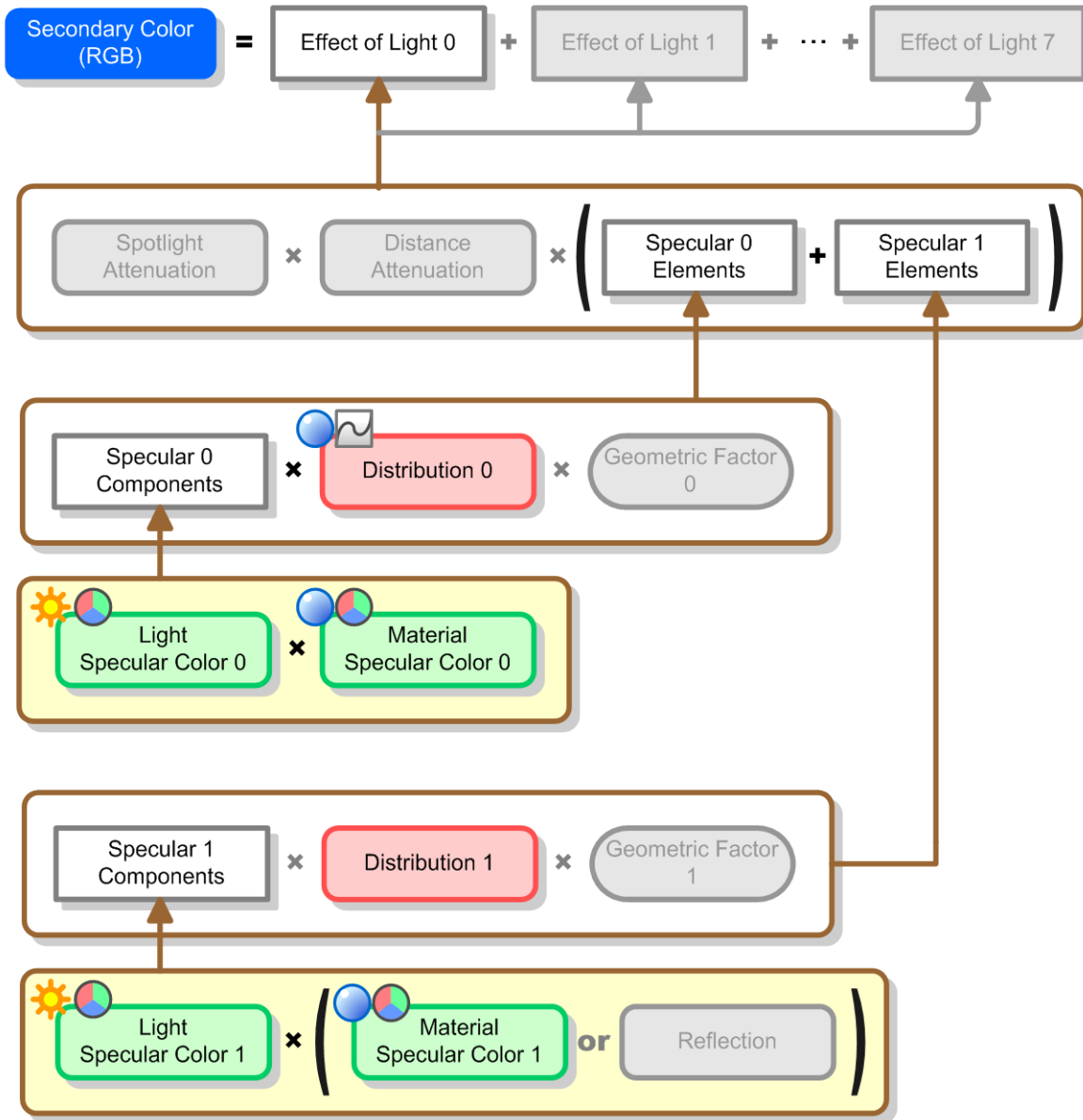


Figure 10-14 Formula for Two-Layer Paint (Secondary Color)



10.1.5 Fresnel Reflection

Fresnel reflection helps depict transparent objects that reflect light, such as glass or water surfaces.

Figure 10-15 Fresnel Reflection



Fresnel reflections express light diffusion and transparency that changes depending on the angle of the viewpoint relative to the object surface, and thus use lookup tables that you configure.

Figure 10-16 Fresnel Reflection Settings

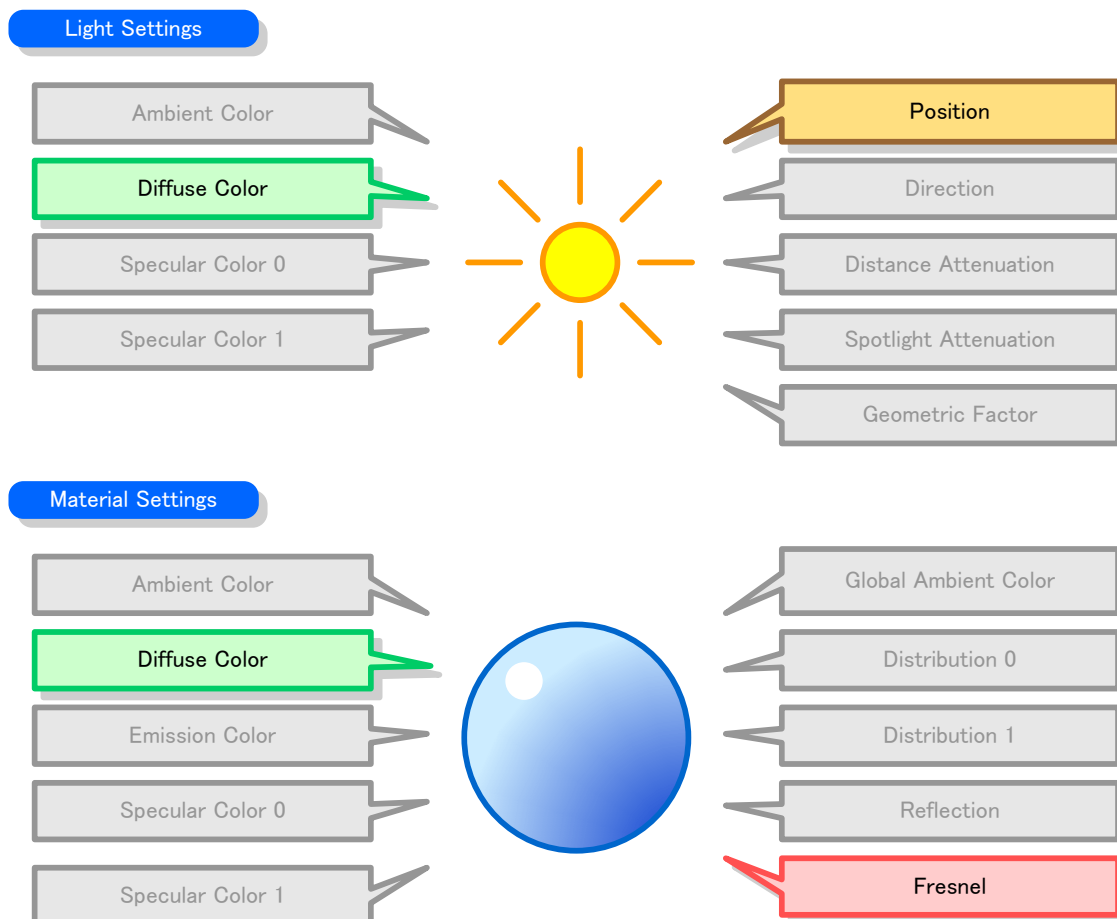
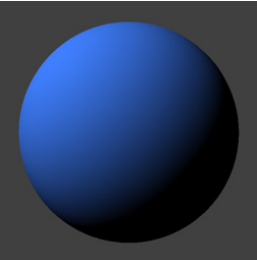
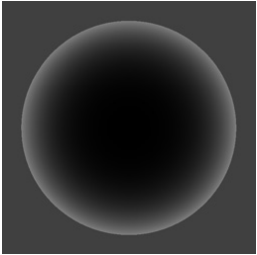
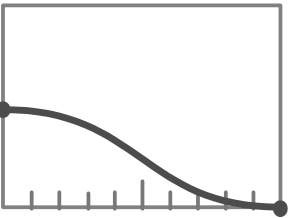
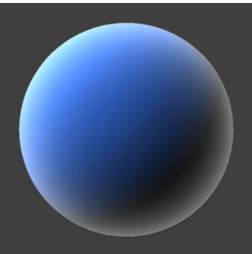
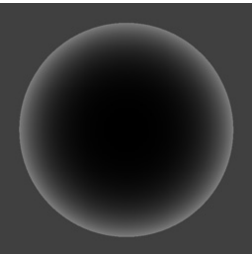


Table 10-7 Sample Settings for Fresnel Reflection (Primary Color)

Setting For	Setting	Setting Example	Lighting Results
Light 	Diffuse Color		RGB 
	Position	Any coordinate	
Material 	Diffuse Color		A 
	Fresnel (Input)	NV	
	Fresnel (Lookup Table)		

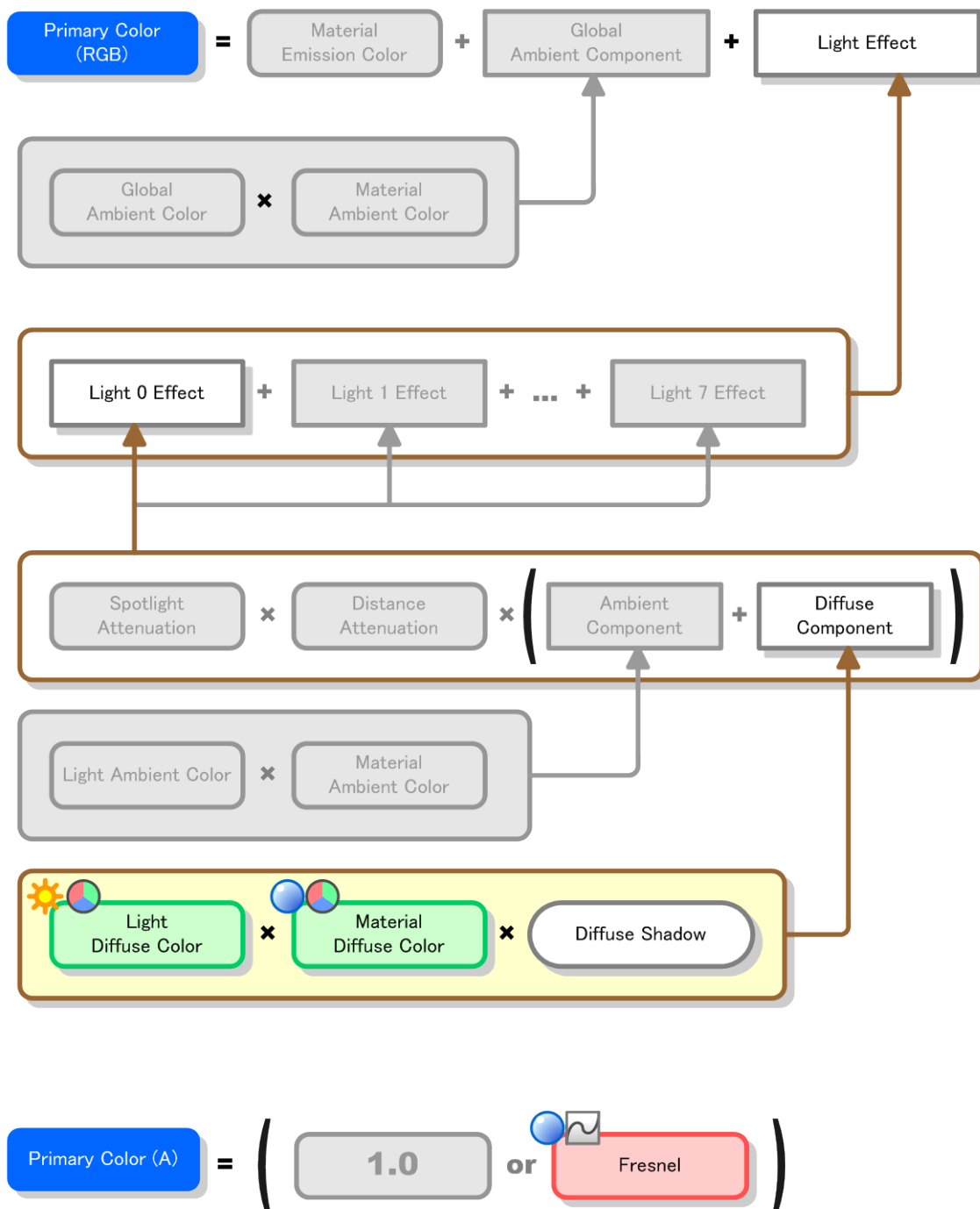
In this settings example, the Fresnel output is not just used for the alpha, it is also added to the color by the texture combiner. The texture combiner settings are as follows.

Table 10-8 Fresnel Reflection Texture Combiner Settings

Combiner	Target	Input Source	Operand	Function	Scale	Output Image
Combiner 0	RGB	Primary Color	RGB	ADD	1.0	
		Primary Color	A			
		-	-			
	A	Primary Color	A	REPLACE	1.0	
		-	-			
		-	-			

The following figure shows the algorithm for calculating Fresnel reflection.

Figure 10-17 Fresnel Reflection Algorithm (Primary Color)



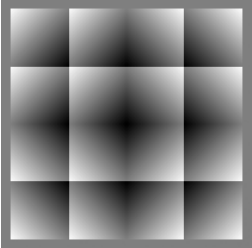
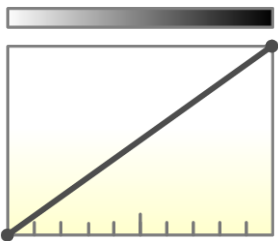
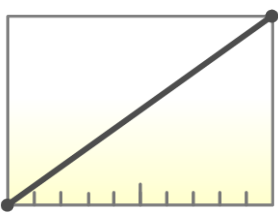
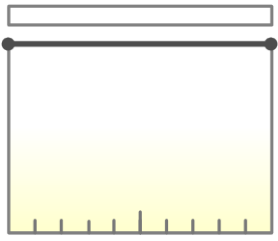
10.2 Sample Settings for Procedural Textures

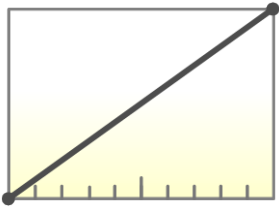
This section gives sample settings for procedural textures.

For more details on each of the procedural texture settings, see section 5.8 Procedural Textures.

Each setting is assumed to be initialized as shown in the following table; if a setting is not mentioned, it is assumed to be unchanged.

Table 10-9 Basic Procedural Texture Settings

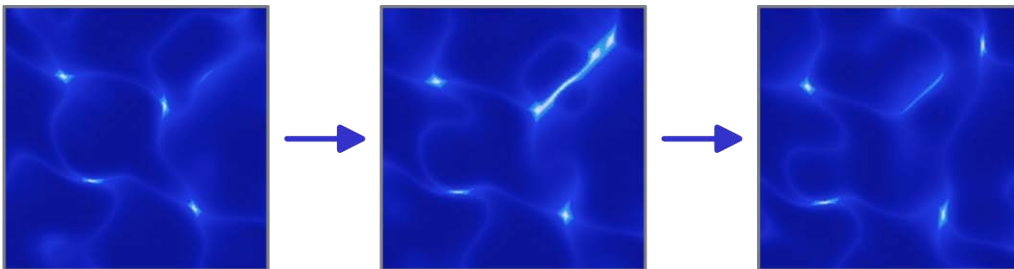
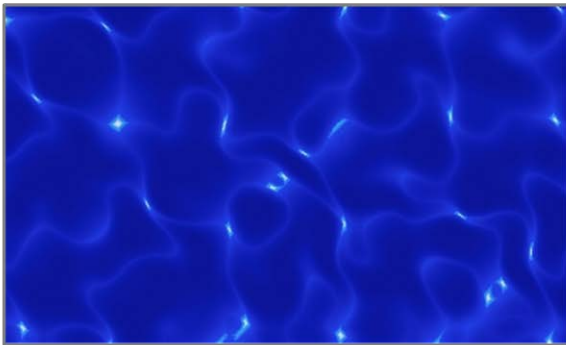
Item		Setting	Image
Tiling	Repeat method (u, v)	(Symmetric, symmetric)	
	Shift (u, v)	(NONE, NONE)	
Basic shape	G function	ADD	
	F function		
Color	RGB		
	A		
	Frequency (u, v)	(0, 0)	
Noise	Phase (u, v)	(0, 0)	
	Amplitude (u, v)	(0, 0)	

Item		Setting	Image
	Noise function		

10.2.1 Water Surfaces

Let's create a procedural texture that resembles the surface of water. By changing the phase settings, you can animate the surface of the water so that it appears to be quivering.

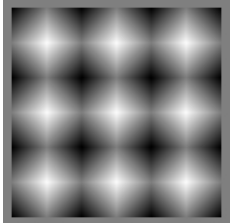
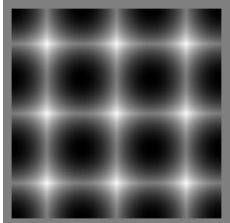
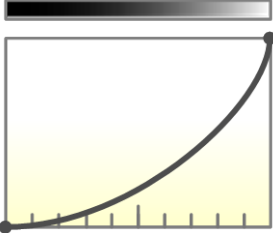
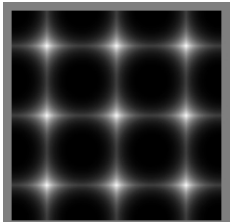
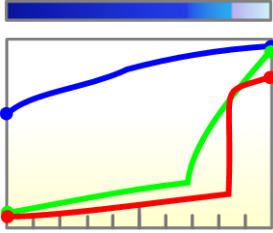
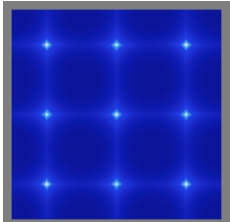
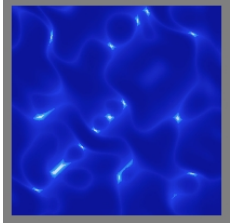
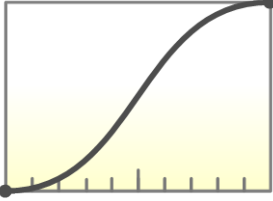
Figure 10-18 Water Surface



You can animate this by changing parameters

The following table shows the settings for a procedural texture that resembles a water surface.

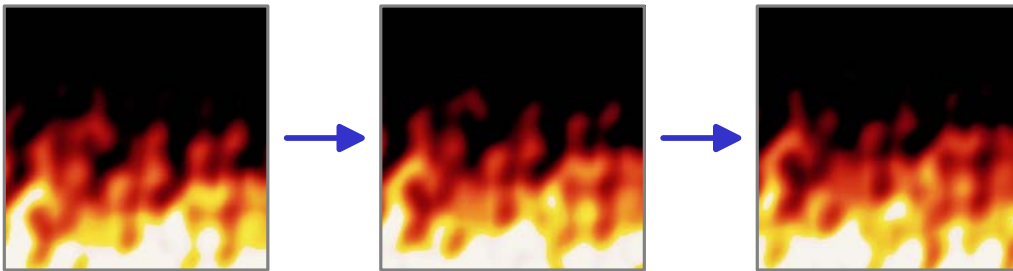
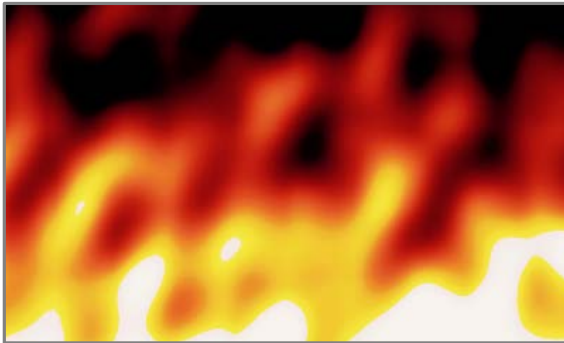
Table 10-10 Sample Settings for the Surface of the Water

Item		Setting	Image
Tiling	Repeat method (u, v)	(mirror, mirror)	(-3.0, 3.0) (3.0, 3.0)  (-3.0, -3.0) (3.0, -3.0)
	Shift (u, v)	(NONE, NONE)	
Basic shape	G function	ADD2	
	F function		
Color	RGB		
Noise	Frequency (u, v)	(0.1, 0.1)	
	Phase (u, v)	(0, 0)	
	Amplitude (u, v)	(2.35, 2.25)	
	Noise function		

10.2.2 Fire

Let's create a procedural texture that resembles fire. By changing the phase settings, you can animate the flames so that they appear to be burning.

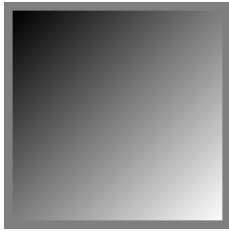
Figure 10-19 Sample Settings for Fire

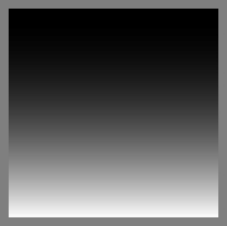
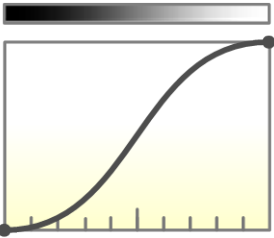
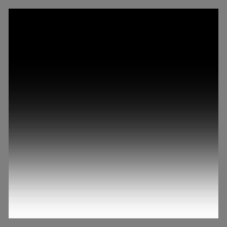
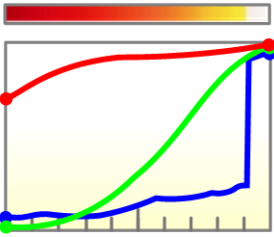
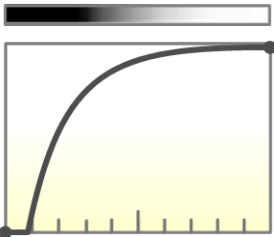
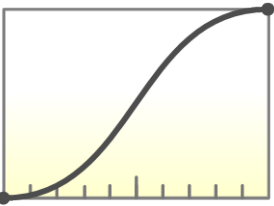


You can animate this by changing parameters

The following table shows the settings for a procedural texture that resembles fire.

Table 10-11 Sample Settings for Fire

Item		Setting	Image
Tiling	Repeat method (u, v)	(mirror, edge clamp)	(0.0, 0.0) (1.0, 0.0) (0.0, -1.0) (1.0, -1.0) 
	Shift (u, v)	(NONE, NONE)	

Item		Setting	Image
Basic shape	G function	V2	
	F function		
Color	RGB		
	A		
Noise	Frequency (u, v)	(1.1, 0.85)	
	Phase (u, v)	(0, 0)	
	Amplitude (u, v)	(1.25, 0.45)	
	Noise function (u, v)		

10.3 Sample Settings for Texture Combiners

This section gives examples of texture combiner configurations that mix textures and lighting results.

For details on each of the texture combiner settings, see Chapter 6 Texture Combiners.

The figures for each sample setting use **A**, **B**, and **C** to represent input sources 0, 1, and 2, respectively, after operands have been specified. Combiners are assumed to have the following settings wherever they have not been explicitly configured.

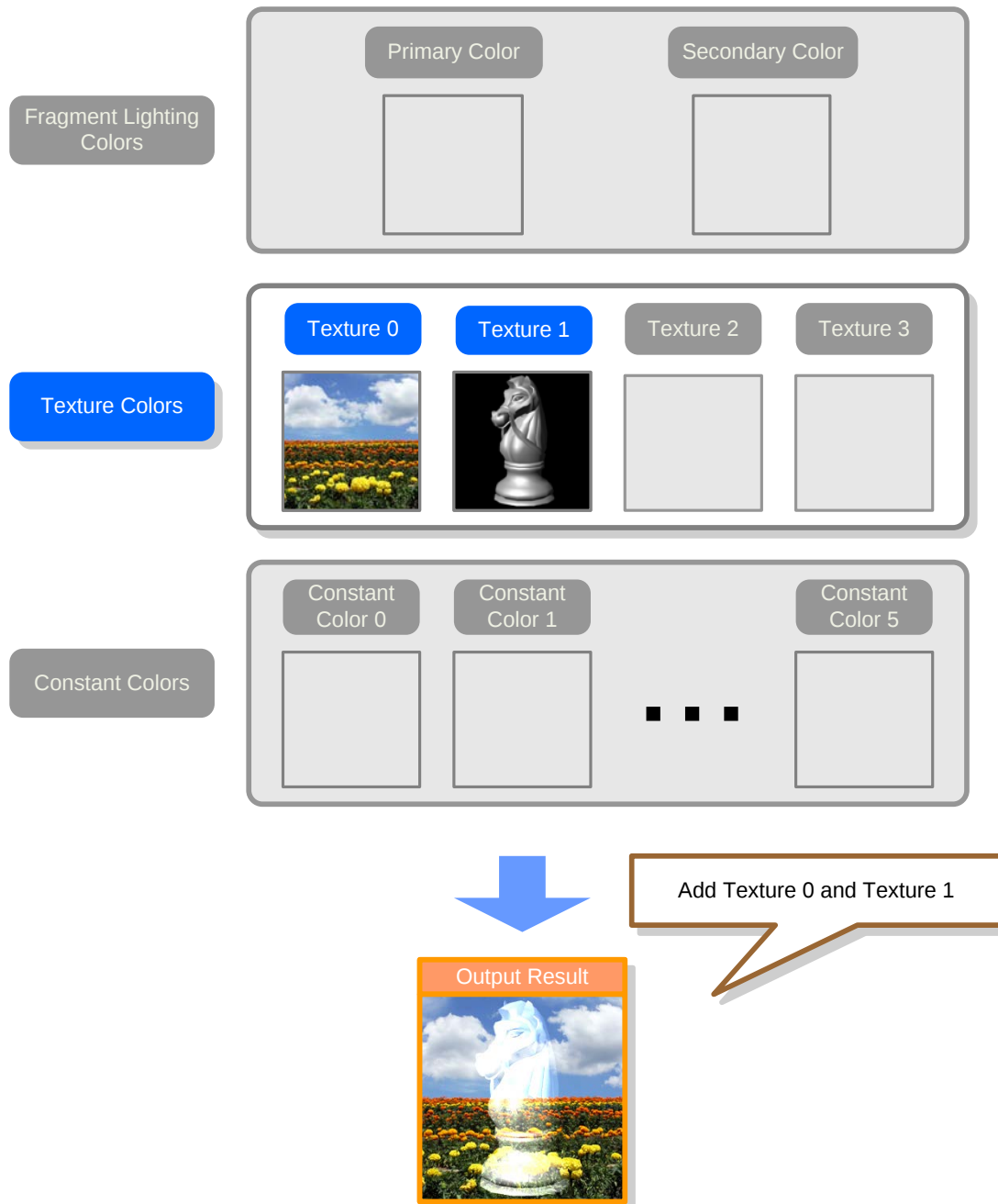
Table 10-12 Settings for Combiners That Have Not Been Explicitly Configured

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Any combiner that is not explicitly configured	RGB	Previous output	RGB	REPLACE	1.0	-
		-	-			
		-	-			
	A	Previous output	A	REPLACE	1.0	-
		-	-			
		-	-			

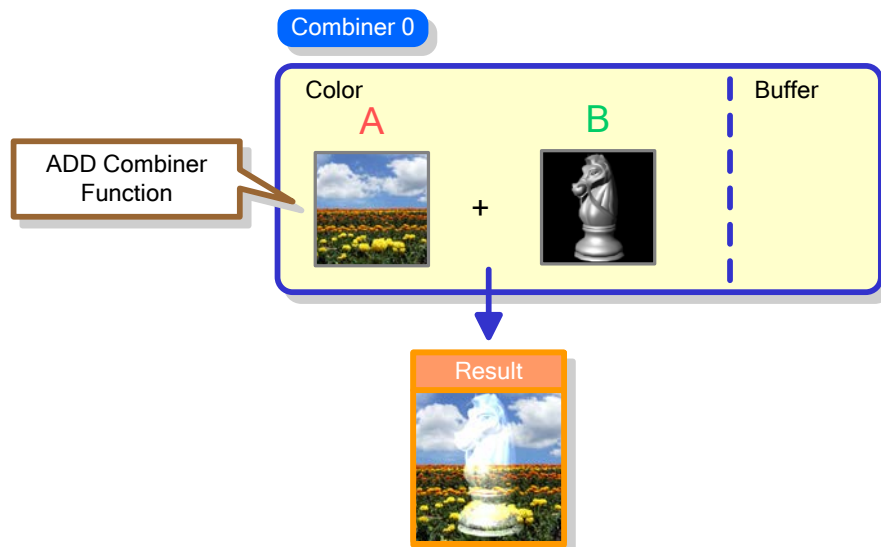
10.3.1 Addition

Adds colors together to combine them. In general, addition results in a brighter image.

Figure 10-20 Sample Addition



This example uses a single texture combiner stage with the ADD combiner function. Addition is processed as follows.

Figure 10-21 Processing for Addition

The following table shows the combiner settings.

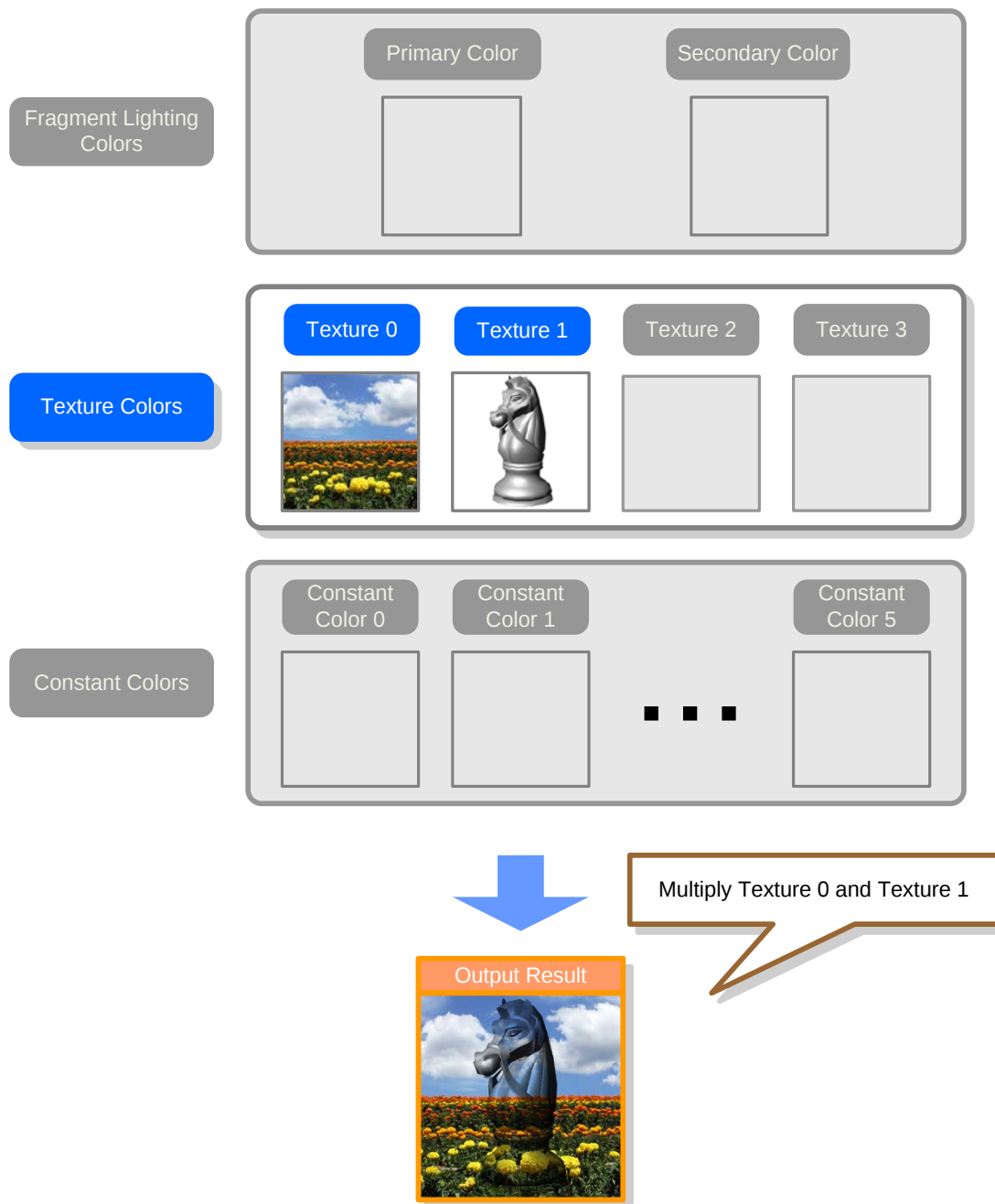
Table 10-13 Settings for Addition

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	RGB	Texture 0	RGB	ADD	1.0	-
		Texture 1	RGB			
		-	-			
	A	-	-	-	-	-
		-	-			
		-	-			

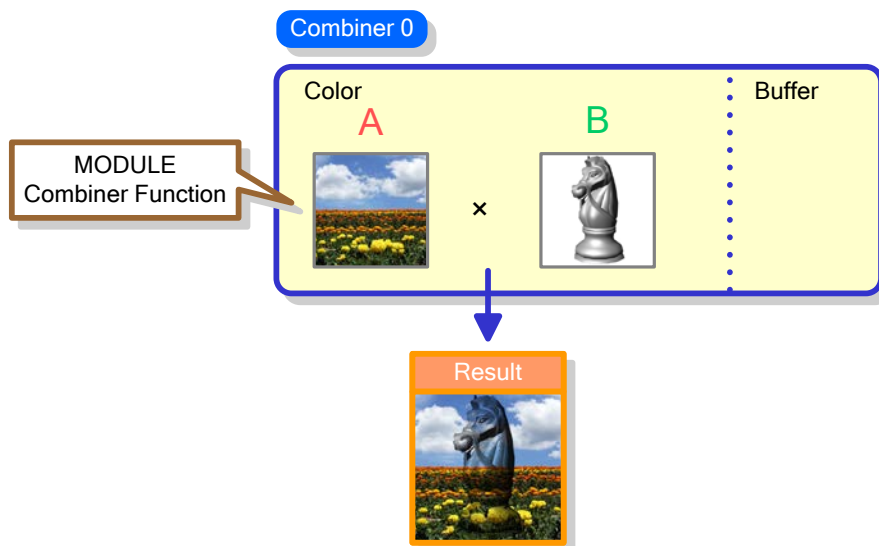
10.3.2 Multiplication

Multiplies two colors together. In general, multiplication results in a darker image.

Figure 10-22 Sample Multiplication



This example uses a single texture combiner stage with the MODULATE combiner function. Multiplication is processed as follows.

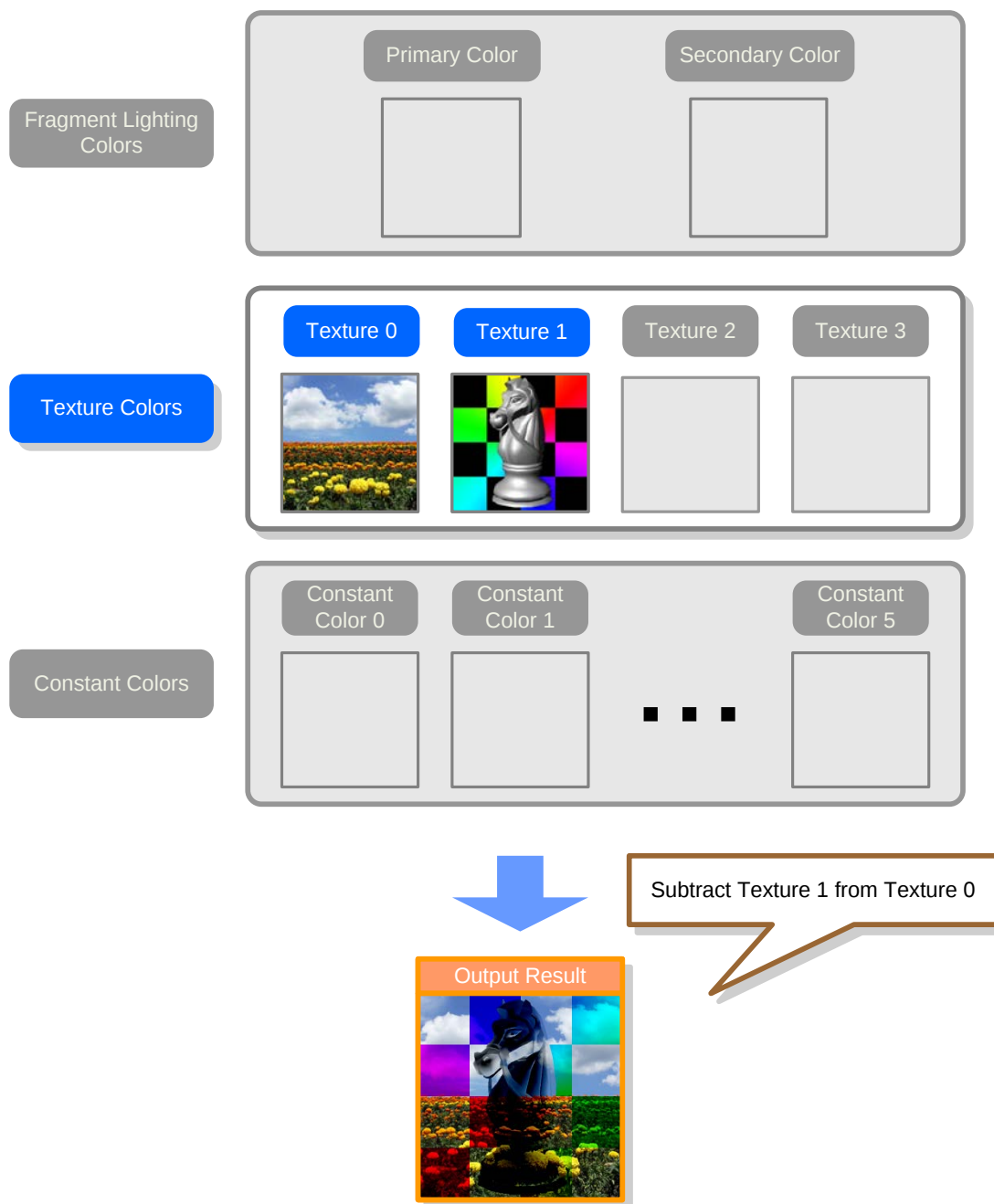
Figure 10-23 Processing for Multiplication**Table 10-14 Settings for Multiplication**

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	RGB	Texture 0	RGB	MODULATE	1.0	-
		Texture 1	RGB			
		-	-			
	A	-	-	-	-	-
		-	-			
		-	-			

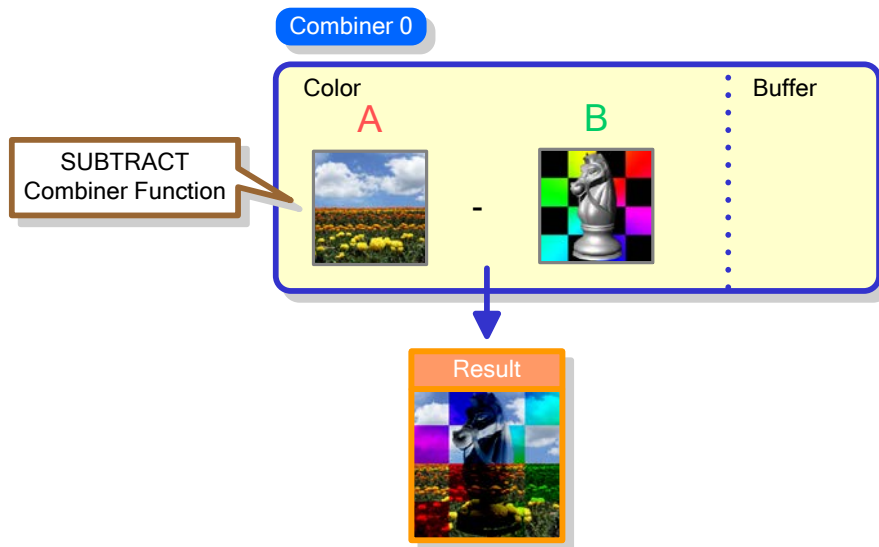
10.3.3 Subtraction

Subtracts one color from another. In general, subtraction results in a darker image.

Figure 10-24 Sample Subtraction



This example uses a single texture combiner stage with the SUBTRACT combiner function. Subtraction is processed as follows.

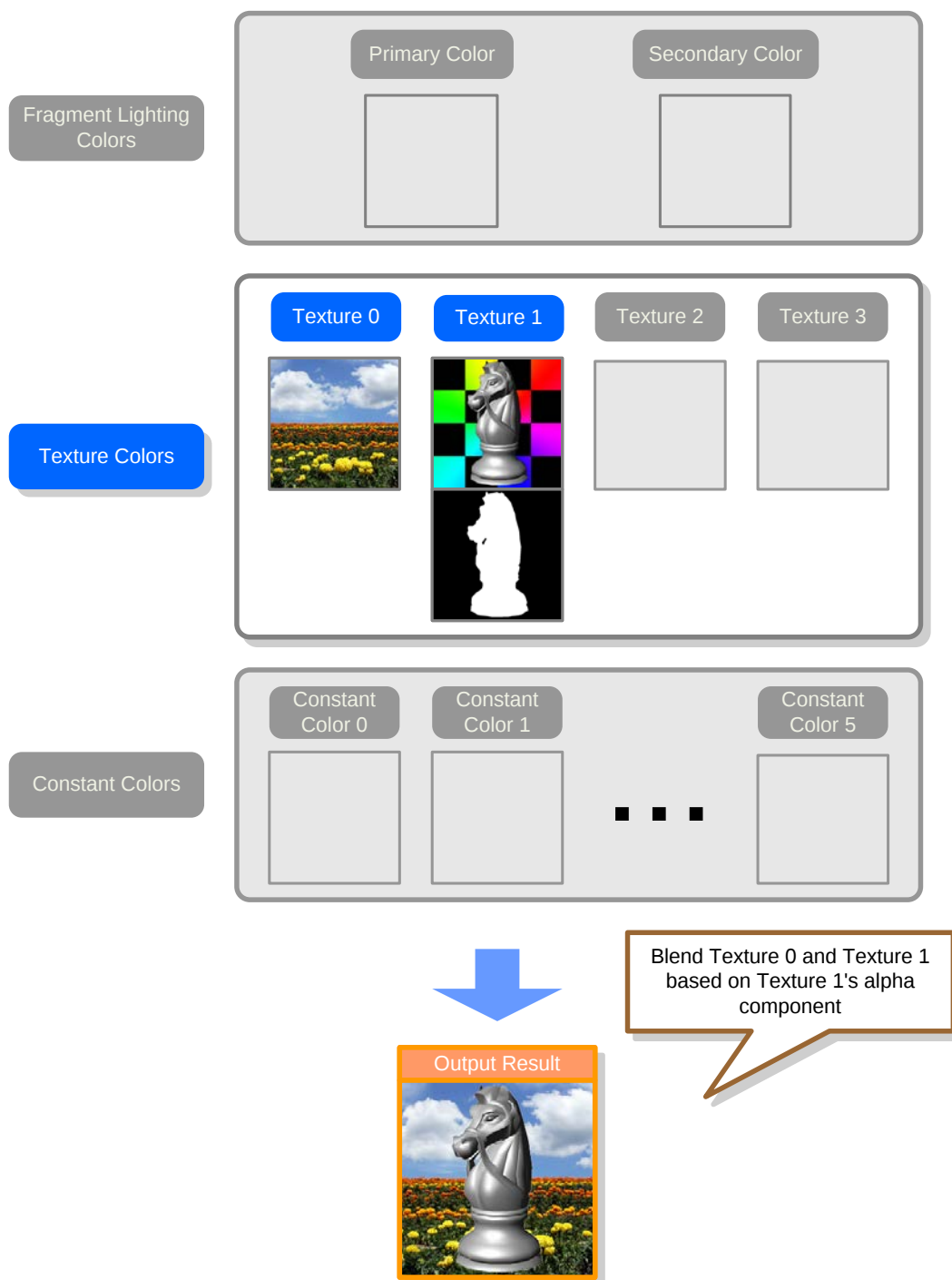
Figure 10-25 Processing for Subtraction**Table 10-15 Settings for Subtraction**

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	RGB	Texture 0	RGB	SUBTRACT	1.0	-
		Texture 1	RGB			
		-	-			
	A	-	-	-	-	-
		-	-			
		-	-			

10.3.4 Decals

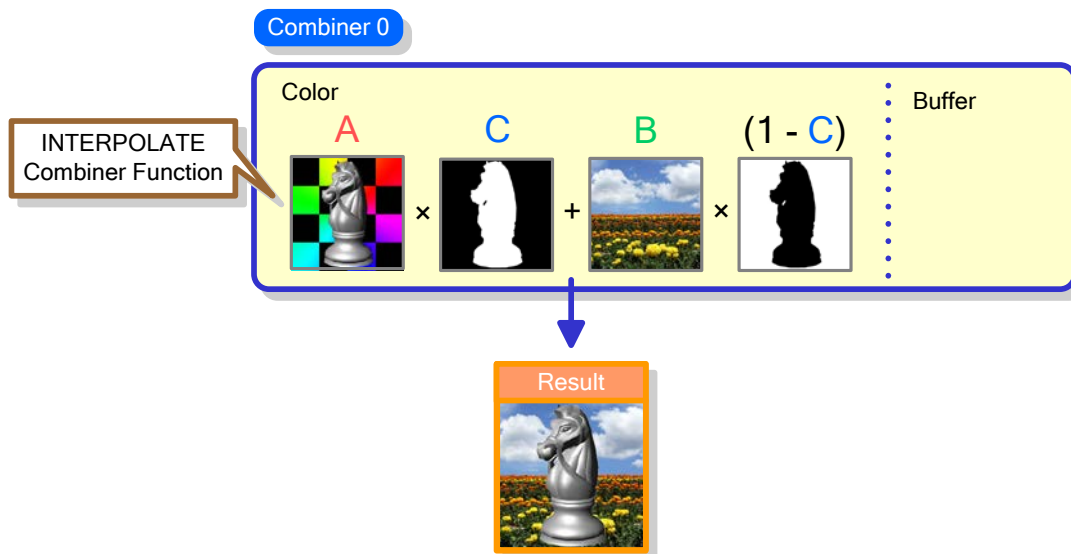
Blends colors together based on a texture with alpha data. This allows you to paste a texture that has been cut out using alpha values onto another texture.

Figure 10-26 Sample Decals



This example uses a single texture combiner stage with the INTERPOLATE combiner function. Decals are processed as follows.

Figure 10-27 Processing for Decals



The following table shows the texture combiner settings.

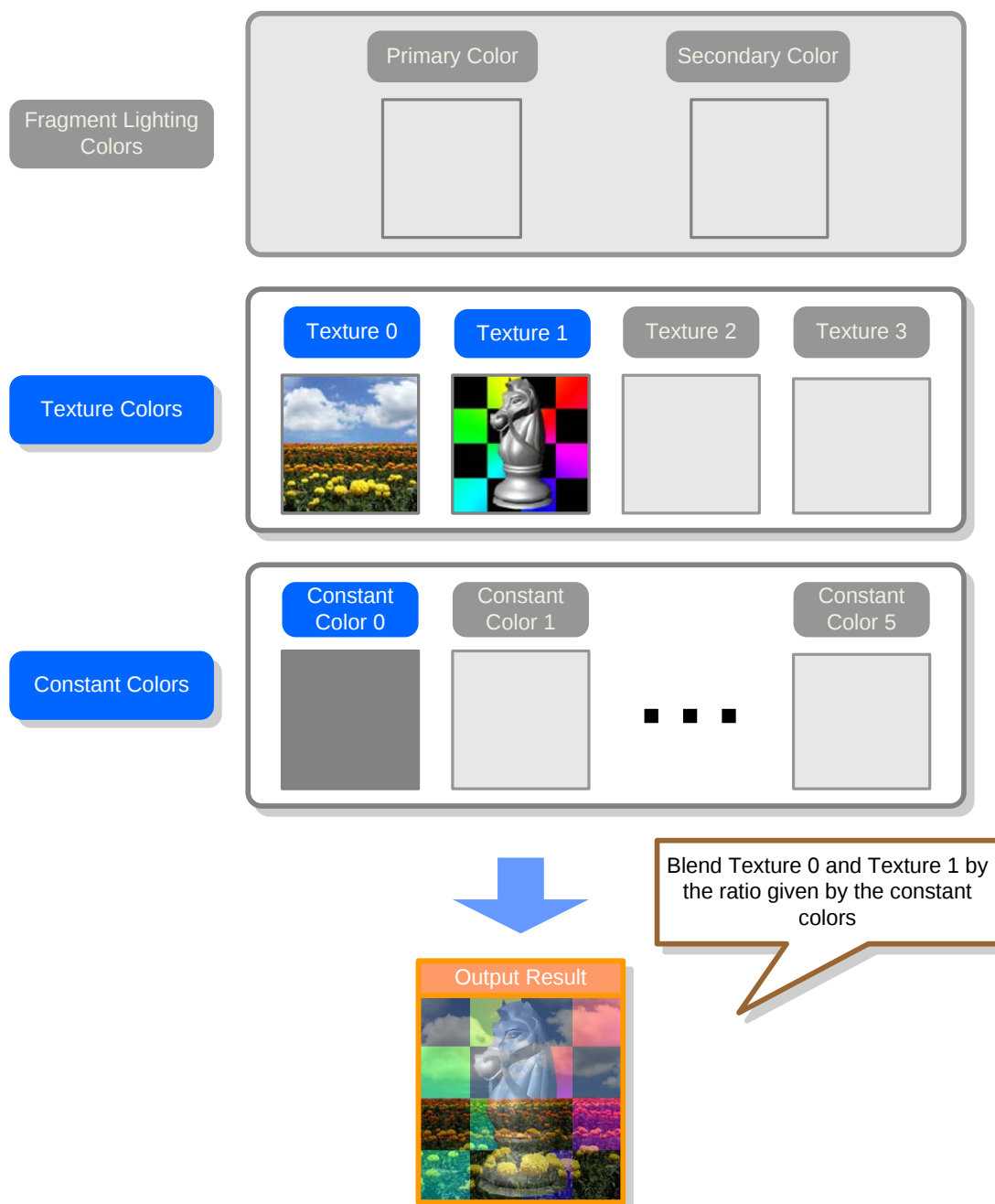
Table 10-16 Settings for Decals

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	Color	Texture 0	RGB	INTERPOLATE	1.0	-
		Texture 1	RGB			
		Texture 1	A			
	Alpha	-	-	-	-	-
		-	-			
		-	-			

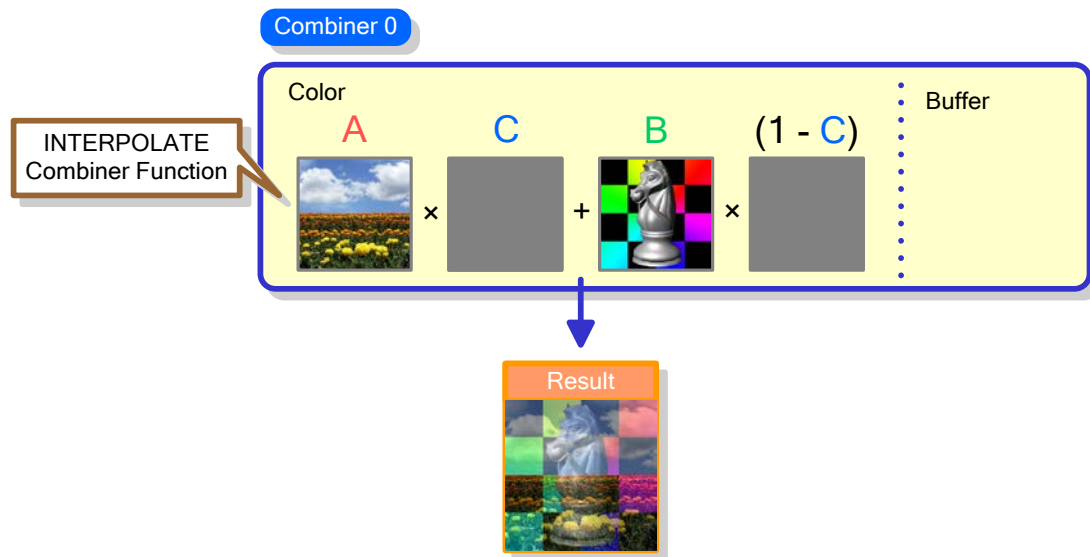
10.3.5 Blending by an Arbitrary Ratio

Blends two colors by whatever ratio has been set. This example uses constant colors to set the ratio. You can slowly change the ratio to show an animation that replaces textures.

Figure 10-28 Example of Blending by a Ratio



This example uses a single texture combiner stage with the INTERPOLATE combiner function. The following figure shows the process of blending by a ratio.

Figure 10-29 Process of Blending by a Ratio

The following table shows the texture combiner settings.

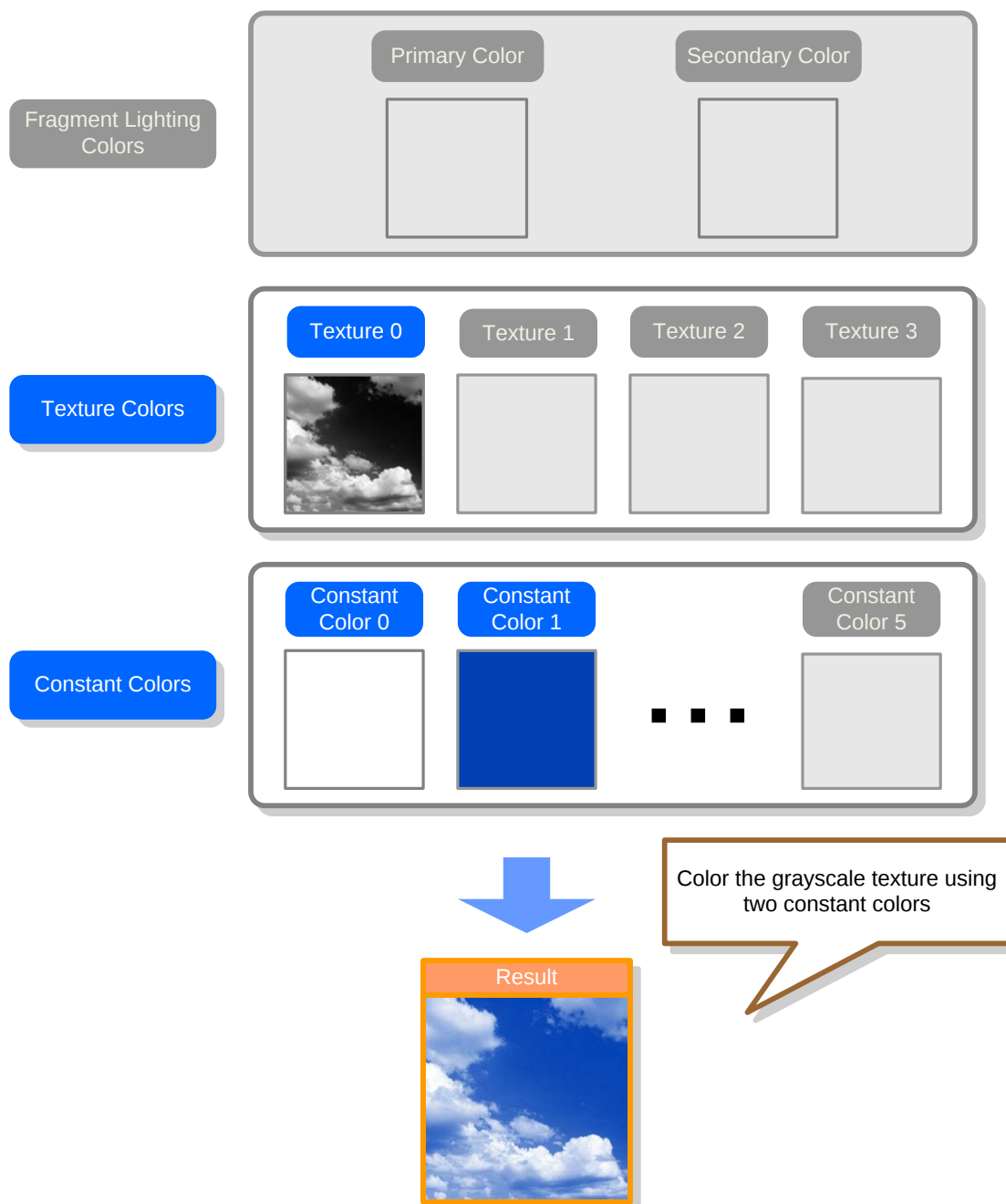
Table 10-17 Settings Used to Blend by a Ratio

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	Color	Texture 0	RGB	INTERPOLATE	1.0	-
		Texture 1	RGB			
		Constant color	RGB			
	Alpha	-	-	-	-	-
		-	-			
		-	-			

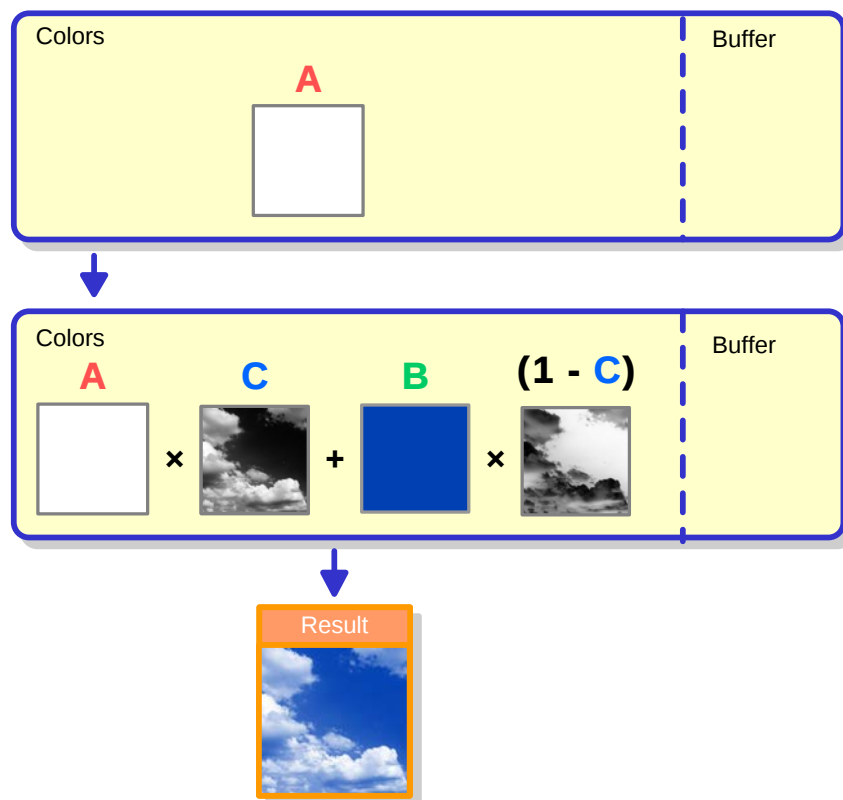
10.3.6 Two-Color Interpolation

Interpolates a grayscale texture between two colors. This example uses two constant colors.

Figure 10-30 Two-Color Interpolation



This example uses two texture combiner stages and sets the combiner functions **REPLACE** and **INTERPOLATE**. Two-color interpolation is processed as follows.

Figure 10-31 Processing Two-Color Interpolation

The following table shows the texture combiner settings.

Table 10-18 Two-Color Interpolation Settings

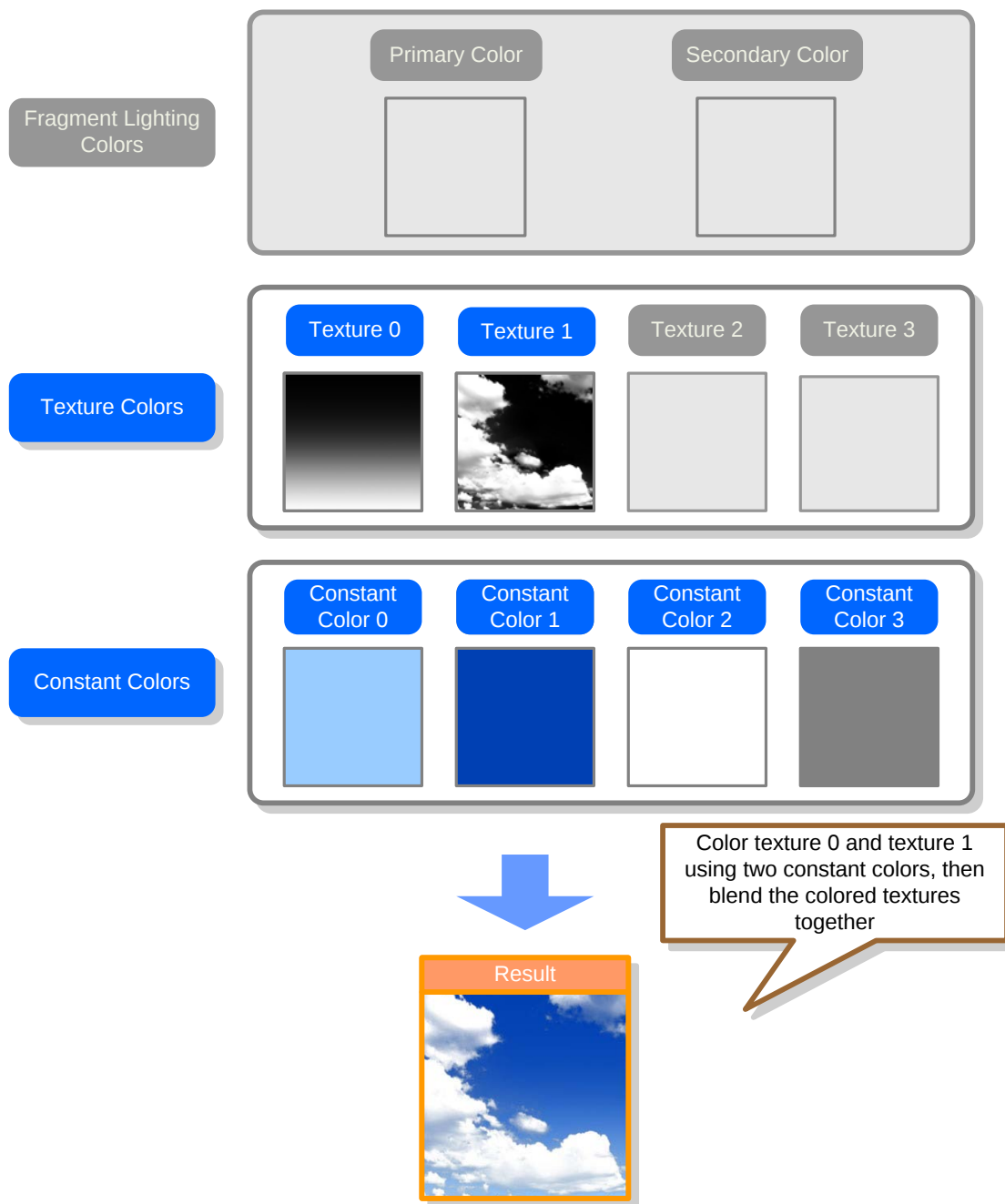
Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	Color	Constant color	RGB	REPLACE	1.0	-
		-	-			
		-	-			
	Alpha	-	-	-	-	-
		-	-			
		-	-			
Combiner 1	Color	Previous output	RGB	INTERPOLATE	1.0	-

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
		Constant color	RGB			
		Texture 1	RGB			
	Alpha	-	-	-	-	-
		-	-			
		-	-			

10.3.7 Applied Two-Color Interpolation

This example interpolates a simple gradation texture and a cloud texture between two constant colors to produce the combined sky texture that is used. By changing the constant colors, you can alter the coloring without changing the texture data and thus represent the sky at different times.

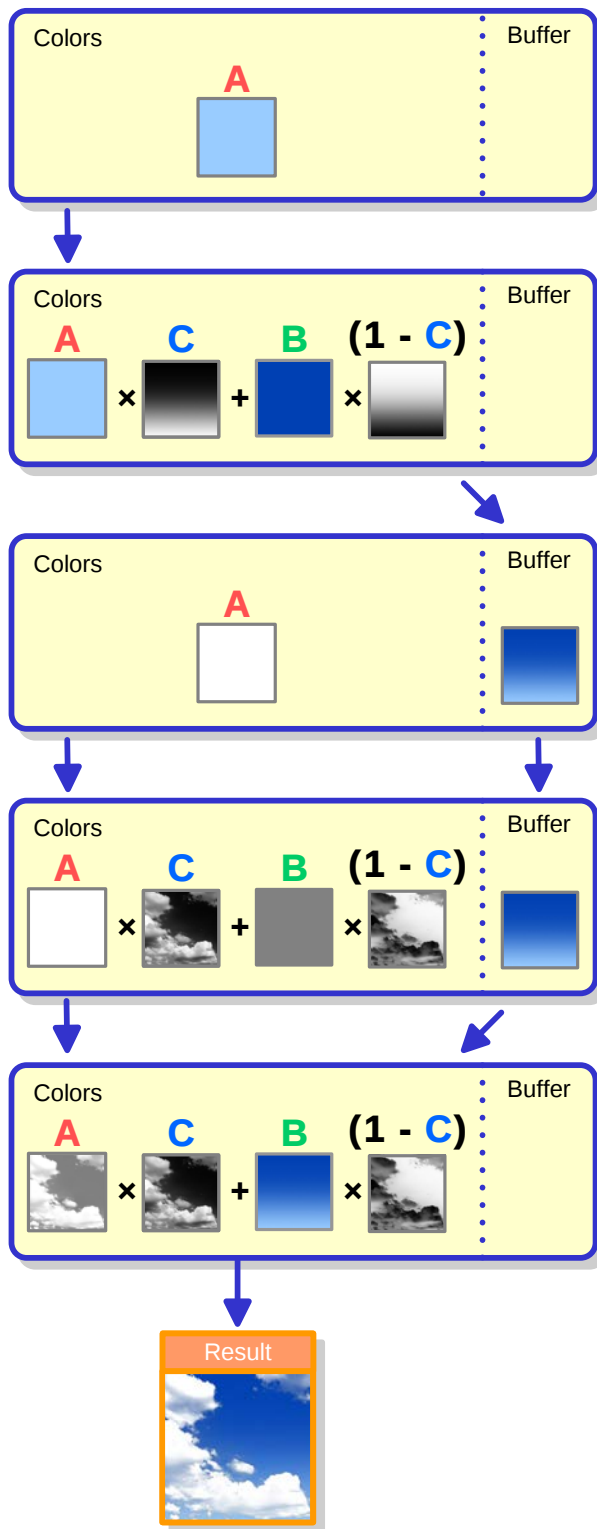
Figure 10-32 Applied Two-Color Interpolation



Note: This example uses a texture to represent a gradation, but vertex colors can also be used to apply a gradation.

This example uses five texture combiner stages and sets the combiner functions **REPLACE** and **INTERPOLATE**. A combiner buffer is used to temporarily store the colors mixed in combiner 1. The following figure shows how the color interpolation is processed.

Figure 10-33 Processing the Applied Two-Color Interpolation



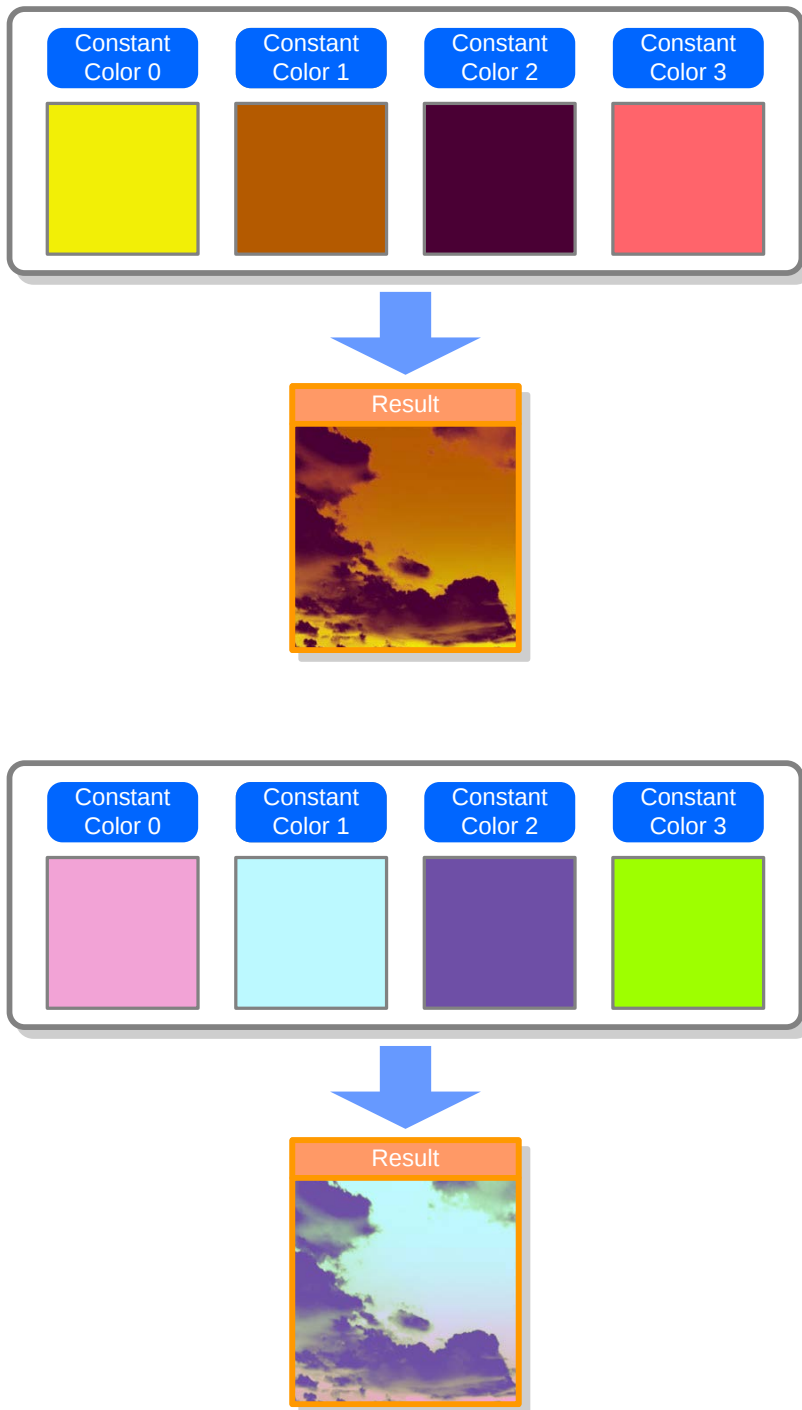
The following table shows the texture combiner settings.

Table 10-19 Settings for Applying Two-Color Interpolation

Combiner	Affected Components	Input Source	Operands	Function	Scale	Combiner Buffer
Combiner 0	Color	Constant color	RGB	REPLACE	1.0	-
		-	-			
		-	-			
	Alpha	-	-	-	-	-
		-	-			
		-	-			
Combiner 1	Color	Previous output	RGB	INTERPOLATE	1.0	-
		Constant color	RGB			
		Texture 0	RGB			
	Alpha	-	-	-	-	-
		-	-			
		-	-			
Combiner 2	Color	Constant color	RGB	REPLACE	1.0	Previous output
		-	-			
		-	-			
	Alpha	-	-	-	-	-
		-	-			
		-	-			
Combiner 3	Color	Previous output	RGB	INTERPOLATE	1.0	Previous combiner buffer
		Constant color	RGB			
		Texture 1	RGB			
	Alpha	-	-	-	-	-
		-	-			
		-	-			
Combiner 4	Color	Previous output	RGB	INTERPOLATE	1.0	-
		Previous buffer	RGB			
		Texture 1	RGB			
	Alpha	-	-	-	-	-
		-	-			
		-	-			

You can change the input constant colors to change the overall coloring.

Figure 10-34 Examples With Changed Constant Colors

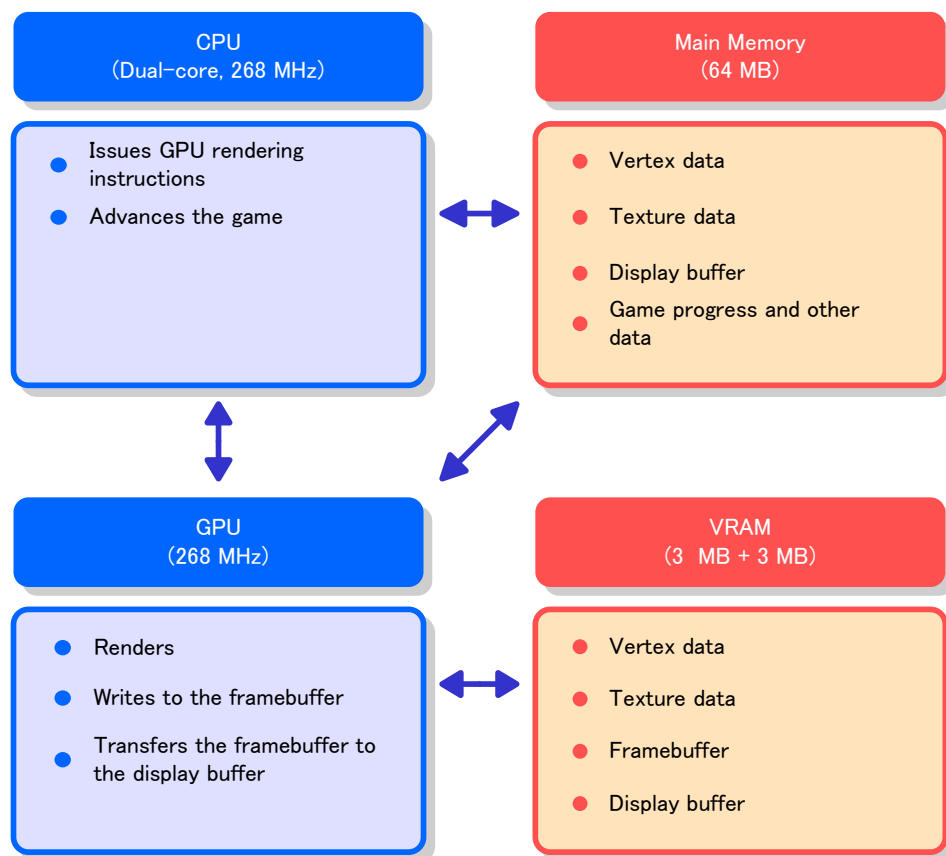


Appendix A

A.1 Hardware Configuration

The following figure shows how hardware related to CTR graphics is organized.

Figure A-1 Hardware Configuration



A.1.1 CPU

The CPU handles processing used to advance the game. This includes animations, 3D model postures, and physical calculations. This also issues rendering commands to the GPU. Two 268 MHz dual-core CPUs are installed on the CTR system. The system uses one CPU core and games use another.

A.1.2 GPU

The GPU handles the actual rendering of polygons and other objects. The rendering process follows the rendering commands issued by the CPU. A 268 MHz GPU is installed on the CTR system.

A.1.3 Main Memory

The following graphics-related data can be placed in main memory: vertex data, texture data, and display buffers. Other data, used for game progress or otherwise, can also be placed here. The CTR system has 64 MB of main memory that can be used by games.

A.1.4 VRAM

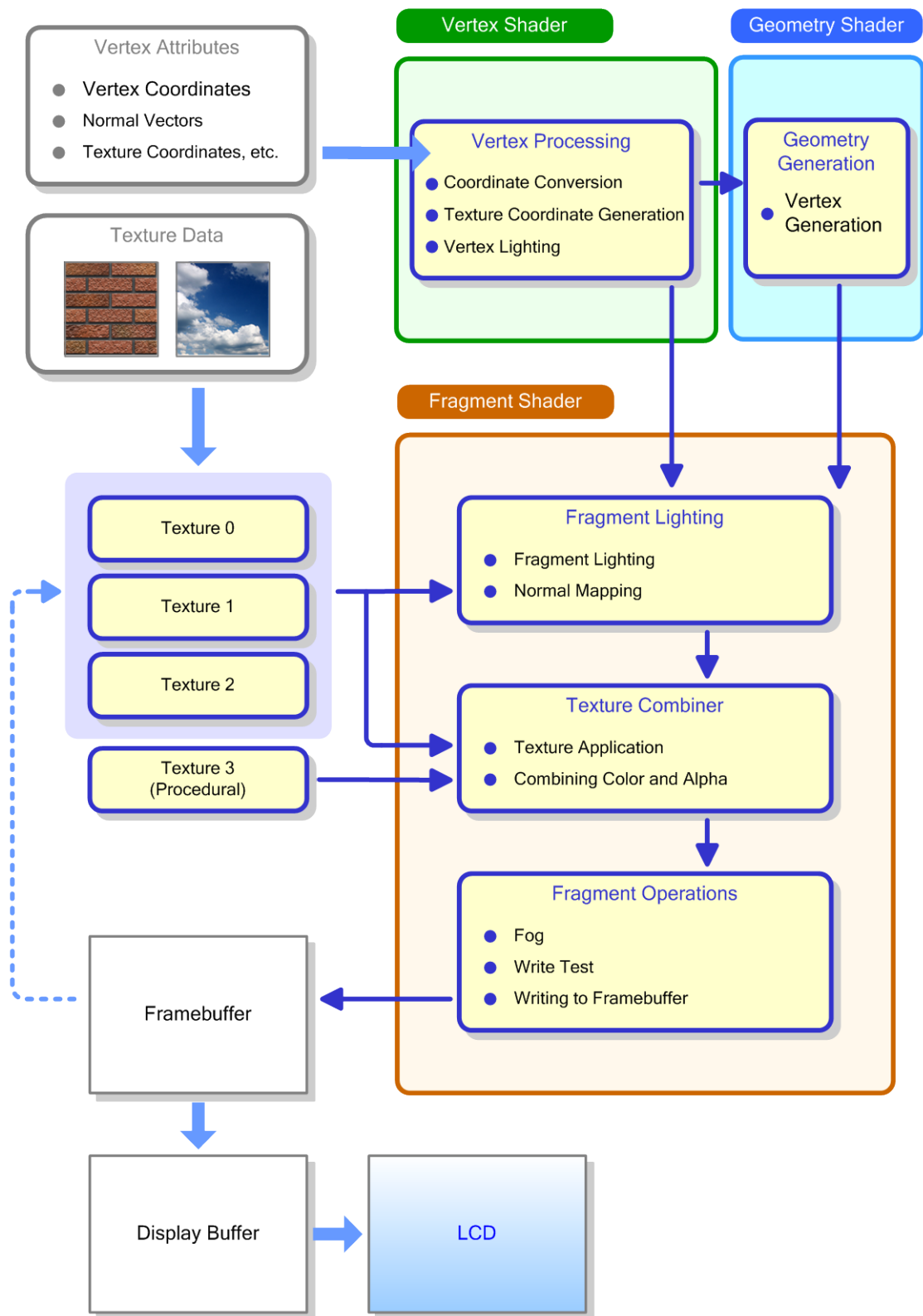
VRAM is a type of memory that is specialized for graphics. Vertex data, texture data, and framebuffers can be placed here. The CTR system has two 3-MB VRAM banks, for a total of 6 MB, that can be used by games.

Because VRAM runs faster than main memory, it is good for placing vertex and texture data that is used often.

A.2 Overall Image of the Graphics Pipeline

The following figure gives an overall picture of the graphics pipeline.

Figure A-2 Overall Image of the Graphics Pipeline



A.3 Performance

This section gives factors that you must be careful of when creating CTR graphics because they affect the processing load. See the appropriate sections for details.

A.3.1 Vertex Processing

- Subdivision levels

The processing load on the geometry shader increases with the subdivision level. For details, see section 3.4.1.1 Levels.

A.3.2 Fragment Lighting

- Number of lights

The cost to process fragment lighting increases with the number of lights used. For more details on this setting, see section 4.3.1 Number of Lights.

- Layer configuration

The layer configuration that you use changes the cost to process lighting. For more details on this setting, see section 4.7 Layer Configurations.

A.3.3 Textures

- Texture size

The cost of processing a texture increases with its size. For more details on this setting, see section 5.1 Texture Size.

- Texture format

The ETC format entails the lowest processing cost and is the fastest. The cost of processing other formats increases with the number of bits per texel. For more details on this setting, see section 5.2.1 Types of Texture Formats.

- Mipmaps

You can use mipmaps to reduce the processing load. For more details on this setting, see section 5.5 Mipmaps.

- Texture filters

Specifying either filtering method within the same level does not change the processing load during texture magnification. There is a higher processing cost associated with a filtering method that smoothly interpolates within the same level during texture minification as well as with a filtering method that smoothly interpolates between mipmap levels. For more details on this setting, see section 5.6 Texture Filters.

- Procedural textures

These entail less processing load than normal textures. Processing load is the same for each procedural texture, regardless of different texture settings. Memory is also the same for each. For more details on this setting, see section 5.8 Procedural Textures.

- Texture combinations in use

The processing load increases with the number of textures registered at one time. For more details on this setting, see section 5.9 Usable Texture Combinations.

A.3.4 Texture Combiners

- Texture Combiner Settings

A texture combiner's processing load is always fixed, regardless of its settings. For more details on this setting, see section 6.3 Texture Combiner Settings.

A.3.5 Fragment Operations

- Write tests

Using either the depth or stencil tests increases the processing load. The processing load is the same when both depth and stencil tests enabled and when just one or the other is enabled. For more details on this setting, see section 7.3 Write Tests.

Revision History

Version	Revision Date	Action	Description
0.8	2011/01/31	Added	10.2 Sample Settings for Procedural Textures - Added content specific to the initial values used by the sample settings. 10.2.1 Water - Added this section. 10.2.2 Fire - Added this section.
		Updated	3.4.1.1 Levels - Updated Figure 3-10 Levels. 3.4.3 Particle System - Added information on particle systems. 5.2.1 Types of Texture Formats - Added information on hardware restrictions. 5.9 Usable Texture Combinations - Added Table 5-20 Combinations of Textures and Texture Coordinates along with information on how combinations of texture coordinates are restricted. 8.3 Antialiasing - Updated Figure 8-2 Antialiasing. Revision History - Moved to end of document.
0.7	2010/10/21		3.4 Geometry Generation - Added descriptions of subdivision and silhouette generation. 4.5.1 Primary Color - Added a note on the conditions under which the material emission color and global ambient elements are valid. 5.2.1.8 HILO - Revised the image in Table 5-9 HILO to have a B component of 0. 7.3.1 Alpha Test - Added Figure 7-5 Sample Usage of the Alpha Test. 7.3.2 Depth Test - Added Figure 7-7 Sample Usage of the Depth Test. 7.3.3 Stencil Test - Changed the caption for and content of Figure 7-9 Sample Usage of the Stencil Test. A.2 Overall Image of the Graphics Pipeline - Added the geometry shader to Figure A-2 Overall Image of the Graphics Pipeline. A.3.1 Vertex Processing - Added this section.

Version	Revision Date	Action	Description
0.6	2010/08/18		- Added section 10.1.5 Fresnel Reflection. 2.2.1 Fragment Shader Effects Changed “Phong seeding” to “fragment lighting.” Changed fragment lighting references. Updated fragment shader figures. 4.6.1 Lookup Table Input Values Revised description of input value SP in Table 4 5 Lookup Table Input Values. Added Table 4 6 Settings Using Lookup Tables and Input Value. 4.7 Layer Configurations Revised processor load for layer configurations 4, 5, and 5 from “x 2” to “x 3” in Table 4 7 Layer Configurations. Added “Distance Attenuation” column. 5.2.1 Types of Texture Formats Added “Minimum Size” column to Table 5 1 Types of Texture Formats. 5.7.4 Normal Mapping Inserted addendum about flipping normal mapping. 5.8 Procedural Textures Moved from 5.4, added content. 6.3.1 Input Sources Revised wording about combiner numbers in the notes. 7.2 Fog Updated Figure 7 3 Fog. 7.3 Write Tests Revised description of processing load for stencil and depth tests. - A.1 Hardware Configuration Updated VRAM capacity description from “x 2” to “x 3.” Updated VRAM capacity and changed colors in Figure A 1 Hardware Configuration. - A.2 Overall Image of the Graphics Pipeline Revised text. Updated Figure A 2 Overall Image of the Graphics Pipeline. - A.3.2 Textures Added description of procedural textures. - A.3.4 Fragment Operations Revised description of processing load for stencil and depth tests.
0.5	2010/06/10		- Added section 10.3.6 Two-Color Interpolation. - Added section 10.3.7 Applied Two-Color Interpolation. - Added Figure A-2 Overall Image of the Graphics Pipeline. - Revised the wording of a note in section 4.2 Fragment Lighting Overview. - Updated Figure 4 8 Distance Attenuation. - Removed layer configurations 8, 9, and 10 from section 4.7 Layer Configurations and also changed the processing load for layer configuration 7 from x3 to x4. - Removed figures from Table 5 11 Alpha Image List for Texture Formats. - Updated notes related to the processing load for filters during texture minification in section 5.6 Texture Filters. - Updated Figure 5 13 UV Mapping. - Updated Figure 5 14 Projective Mapping.

Version	Revision Date	Action	Description
			• Updated Figure 5 15 Cube Environment Mapping. • Updated Figure 5 16 Normal Mapping. • Revised the types of write tests that increase the processing load in section 7.3 Write Tests. • Added combiner settings to section 10.3 Sample Settings for Texture Combiners; these apply to combiners that are not explicitly configured. Also changed the format of the processing figures and each of the sample settings. • Updated section A.3.2 Textures with descriptions of the processing load when filters are used for texture minification . • Revised the types of write tests that increase the processing load in section A.3.4 Fragment Operations.
0.4	2010/04/22		• Added section 10.1.4 Two-Layer Paint. • Added information to section 2.1 CTR Screens on the autostereoscopic LCD. • Revised text and updated figures in section 4.3.5 Distance Attenuation. • Updated Figure 4-9 Spotlight Attenuation in section 4.3.6 Spotlight Attenuation. • Revised vector descriptions in Figure 4-25 SP Input Value and the text of section 4.6.1.5 SP Input Value. • Updated Figure 5-5 Cube Map Textures in section 5.3 Cube Map Textures. • Updated Figure 5-15 Cube Environment Mapping in section 5.8.3 Cube Environment Mapping. • Added supplementary information on the order of tests and on the processing load in section 7.3 Write Tests. • Updated section A.1 Hardware Configuration. • Updated section A.3 Performance.
0.3	2010/03/18		• Added section 4.3.3 Spotlight Direction and a figure. • Made a separate section, 4.6.1 Lookup Table Input Values, and added descriptions of each input value. • Added section 10.1.3 Toon to the sample settings for fragment lighting in section 10.1 Sample Settings for Fragment Lighting. • Changed the order of settings in Figure 4-4 Light Settings in section 4.3 Light Settings. • Modified the type and order of light sources in Figure 4-5 Types of Light Sources. • Renamed section 4.3.2 Position from "Position and Direction". Added descriptions and figures. • Changed the order of sections 4.3.5 Distance Attenuation and 4.3.6 Spotlight Attenuation. • Updated text in section 4.3.6 Spotlight Attenuation. • Changed the order of table elements in section 4.6 Effects That Use Lookup Tables. • Added supplementary information on the texture size and processing load to section 5.1 Texture Size. • Added supplementary information on the texture format and processing load to section 5.2.1 Types of Texture Formats. • Updated figures in section 5.6 Mipmaps. • Updated texture images in the figure in section 5.6.1 Mipmap Levels. • Updated figures and text in section 5.6.2 LOD Bias.

Version	Revision Date	Action	Description
			• Updated tables and figures in section 5.7 Texture Filters. • Revised supplementary information and text related to textures being used simultaneously in section 5.9 Usable Texture Combinations. • Moved supplementary information from section 6.2 Texture Combiner Structure to section 6.3 Texture Combiner Settings. • Revised texture color descriptions in the table in section 6.3.1 Input Sources. • Fixed an error in the number of levels for lookup table input values, both in the text and figure in section 7.2 Fog. • Renamed the figure and revised text in section 10.1.1 Lambert.
0.2	2010/02/19		• Added sections 10.1.1 Lambert and 10.1.2 Phong to the sample settings for fragment lighting in section 10.1 Sample Settings for Fragment Lighting. • Added sections 10.3.1 Addition, 10.3.2 Multiplication, 10.3.3 Subtraction, 10.3.4 Decals, and 10.3.5 Blending by an Arbitrary Ratio to the sample settings for texture combiners in section 10.3 Sample Settings for Texture Combiners. • Revised the normal ordering used to determine the front face of a polygon from clockwise to counter-clockwise in section 3.2 Primitives. • Updated figures in section 4.1 What Is Fragment Lighting?. • Fixed inconsistencies in notation in section 4.3.3.7 Light Colors. • Updated figures and text in section 4.3.6 Geometric Factors. • Fixed inconsistencies in notation in section 4.4.1 Material Colors. • Updated figures in section 4.4.3 Distribution (Specular Shape). • Updated figures in section 4.4.4 Reflection (Reflected Colors). • Updated figures in section 4.4.5 Fresnel (Transparency). • Updated figures in section 4.5.1 Primary Color. • Updated figures in section 4.5.2 Secondary Color. • Updated figures in section 5.1 Texture Size. • Added images and changed the order of tables in section 5.2.2 Image List for Texture Formats. • Updated figures in section 5.2.4 Texture Format Size Comparison. • Updated figures and tables in section 5.5 Tiling Textures. • Updated figures in section 5.6.1 Mipmap Levels. • Updated figures in section 5.8 Texture Mapping. • Added information to section 5.9 Usable Texture Combinations on how the processing load is affected by the number of textures used simultaneously. • Fixed inconsistencies in notation in section 8.1 Framebuffer Formats.
0.1	2010/01/21		Initial version.

All company and product names in this document are the trademarks or registered trademarks of their respective companies.

© 2010–2011 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed, or loaned in whole or in part without the prior approval of Nintendo.